

Alibaba Cloud

MaxCompute
Development

Document Version: 20201014

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 <code>cd /d C:/window</code> command to enter the Windows system folder.
<i>Italic</i>	Italic formatting is used for parameters and variables.	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] or [a b]	This format is used for an optional value, where only one item can be selected.	<code>ipconfig [-all -t]</code>
{ } or {a b}	This format is used for a required value, where only one item can be selected.	<code>switch {active stand}</code>

Table of Contents

1.Common commands	12
1.1. List of common commands	12
1.2. Project operations	13
1.3. Table-level operations	19
1.4. Instance operations	26
1.5. Resource operations	30
1.6. Functions operations	32
1.7. SET operations	34
1.8. Other operations	39
2.Data upload and download	41
2.1. Overview	41
2.2. Connection to data tunnel service	42
2.3. Selection of tools to migrate data to the cloud	42
2.4. Introduction to the tools	43
2.5. Tunnel SDK	44
2.5.1. MaxCompute Tunnel overview	45
2.5.2. TableTunnel	46
2.5.3. InstanceTunnel	47
2.5.4. UploadSession	48
2.5.5. DownloadSession	50
2.5.6. TunnelBufferedWriter	52
2.6. Bulk data channel SDK example	53
2.6.1. Example	53
2.6.2. Simple upload	53
2.6.3. Simple download	55
2.6.4. Example for multi-thread uploading	60

2.6.5. Example for multi-thread downloading	63
2.6.6. Example for BufferedWriter uploading	67
2.6.7. Example for BufferedWriter multi-thread uploading	68
2.7. Import or export data by using Data Integration	69
3. SQL	71
3.1. SQL overview	71
3.2. MaxCompute SQL limits	72
3.3. Differences with other SQL statements	74
3.4. Operators	78
3.5. DDL SQL	82
3.5.1. Table-level operations	82
3.5.2. Lifecycle management operations	96
3.5.3. Partition and column operations	98
3.5.4. View-related operations	104
3.6. Insert Operation	106
3.6.1. MULTI INSERT	106
3.6.2. Insert data in dynamic partition mode (DYNAMIC PA...	107
3.6.3. VALUES	110
3.7. Lateral View	114
3.8. Select Operation	116
3.8.1. SELECT syntax	116
3.8.2. SELECT Sequence	121
3.8.3. UNION, INTERSECT, and EXCEPT	123
3.8.4. JOIN	126
3.8.5. SEMI JOIN	129
3.8.6. MAPJOIN HINT	130
3.8.7. HAVING clause	132
3.8.8. Grouping Sets	132

3.8.9. SELECT TRANSFORM	135
3.9. Script Mode SQL	141
3.10. Common table expression (CTE)	145
3.11. Explain	146
3.12. LOAD	149
3.13. DDL SQL	153
3.14. Insert Operation	153
3.15. Builtin functions	153
3.15.1. Comparison of functions built in MaxCompute, MySQL... ..	153
3.15.2. Date functions	185
3.15.3. Mathematical functions	200
3.15.4. Window functions	219
3.15.5. Aggregate functions	250
3.15.6. String functions	260
3.15.7. Other functions	284
3.16. UDF	308
3.16.1. Overview	308
3.16.2. Java UDF	309
3.16.3. Python 2 UDFs	325
3.16.4. Python 3 UDFs	333
3.16.5. Code-embedded UDFs	340
3.16.6. SQL functions	344
3.16.7. Open source geospatial UDFs	347
3.16.8. UDTF usage	352
3.17. UDT	353
3.17.1. UDT	353
3.17.2. Usage examples	362
3.18. UDJ	365

3.19. Appendix	374
3.19.1. Escape character	374
3.19.2. LIKE usage	375
3.19.3. Regular expression	376
3.19.4. Reserved words and keywords	378
3.19.5. Hive data type mapping table	379
4. MapReduce	381
4.1. Summary	381
4.1.1. MapReduce	381
4.1.2. Extended MapReduce model	383
4.1.3. Open-source MapReduce	383
4.2. Function Introduction	389
4.2.1. Basic concepts	389
4.2.2. Commands	390
4.2.3. Input and output	392
4.2.4. Resources	392
4.2.5. Local run	392
4.3. Program Example	395
4.3.1. WordCount samples	395
4.3.2. MapOnly samples	398
4.3.3. Multi-input and Output	401
4.3.4. Multi-task samples	407
4.3.5. Secondary Sort samples	410
4.3.6. Resource samples	413
4.3.7. Counter samples	416
4.3.8. Grep samples	420
4.3.9. Join samples	424
4.3.10. Sleep samples	428

4.3.11. Unique samples	430
4.3.12. Sort samples	435
4.3.13. Partition samples	437
4.3.14. Pipeline samples	440
4.4. Java SDK	443
4.4.1. Java SDK	443
4.4.2. Overview of compatible versions of the SDK	450
4.4.3. Java Sandbox	468
5.External table	474
5.1. Overview of External tables	474
5.2. STS authorization	475
5.3. Access OSS data by using the built-in extractor	479
5.4. Access unstructured data in OSS by using a custom extr... ..	485
5.5. Partition data by using foreign tables	491
5.6. Process OSS data stored in open-source formats	494
5.7. Export unstructured data to OSS	502
5.8. Access Table Store (OTS) data	512
6.PyODPS	520
6.1. Quick start	520
6.2. Installation guide and limits	522
6.3. Platform instructions	524
6.3.1. Overview	524
6.3.2. Migrate PyODPS nodes from a data development pla... ..	524
6.3.3. Use PyODPS in DataWorks	525
6.3.4. Create a PyODPS 3 node	527
6.4. Basic operations	529
6.4.1. Overview	529
6.4.2. Projects	529

6.4.3. Tables	529
6.4.4. SQL	539
6.4.5. Task instances	544
6.4.6. Resources	546
6.4.7. Functions	548
6.5. Configurations	549
6.6. DataFrame	553
6.6.1. Quick start	553
6.6.2. Create a DataFrame object	559
6.6.3. Sequence	561
6.6.4. Collection	564
6.6.5. Execution	575
6.6.6. Column operations	582
6.6.7. Sort, deduplicate, sample, and transform data	596
6.6.8. Use UDFs and the third-party Python libraries	604
6.6.9. MapReduce API	614
6.6.10. Debugging	623
6.7. User experience enhancement	625
6.7.1. Command line	626
6.7.2. IPython	627
6.7.3. Jupyter Notebook	630
6.8. API overview	631
6.9. Examples	631
6.9.1. Reference a third-party package in a PyODPS node	632
6.9.2. Use a PyODPS node to query data based on specific...	635
6.9.3. Use a PyODPS node to pass parameters	643
6.9.4. Use a PyODPS node to read data from a partitioned...	645
6.9.5. Use a PyODPS node to read data from the level-1 p...	649

6.9.6. Use a PyODPS node to perform sequence operations	651
6.9.7. Use a PyODPS node to aggregate data	657
6.9.8. Use a PyODPS node to perform column operations	664
6.9.9. Use a PyODPS node to sort data	672
6.9.10. Use a PyODPS node to de-duplicate data	675
6.9.11. Use a PyODPS node to sample data	678
6.9.12. Use a PyODPS node to scale data	691
6.9.13. Use a PyODPS node to process NULL values	696
7. Spark	701
7.1. Spark on MaxCompute overview	701
7.2. Set up a Spark on MaxCompute development environme...	702
7.3. Running modes	704
7.4. Java and Scala development examples	705
7.4.1. Develop a Spark on MaxCompute application by usin...	705
7.4.2. Spark-2.x examples	713
7.5. Develop a Spark on MaxCompute application by using P...	719
7.6. Diagnose a Spark on MaxCompute job	723
7.7. Access instances in a VPC from Spark on MaxCompute	725
7.8. Configure Spark on MaxCompute to access OSS resources	734
8. Interactive SQL (Lightning)	737
8.1. Overview	737
8.2. Constraints and limitations	738
8.3. Quick Start	739
8.4. Access domain name	740
8.5. Access services using JDBC interfaces	742
8.5.1. JDBC driver	742
8.5.2. Configure JDBC connections	742
8.5.3. Access services using common tools	743

8.6. Query syntax and built-in functions	747
8.7. View tasks	748
9. Graph	749
9.1. Summary	749
9.2. Aggregator	751
9.3. Limits	759
9.4. Function overview	759
9.5. SDK summary	763
9.6. Development and debugging	764
9.7. Examples	769
9.7.1. SSSP	769
9.7.2. PageRank	773
9.7.3. Kmeans	777
9.7.4. BiPartiteMatching	784
9.7.5. Strongly-connected component	789
9.7.6. Connected component	798
9.7.7. Topology Sorting	802
9.7.8. Linear regression	806
9.7.9. Triangle Count	812
9.7.10. Vertex Input	815
9.7.11. Edge Input	823
10. View Job Running Information	832
10.1. Logview	832
10.2. Use Logview V2.0 to view job running information	834
10.3. Errors and warnings using the MaxCompute compiler	839
11. Open-source features of MaxCompute	841

1.Common commands

1.1. List of common commands

This module explains how to use the relevant commands through the client to help you quickly understand MaxCompute.

The latest MaxCompute service adjusts the usual commands, the new command style is more closely used by hive, which is convenient for original Hadoop/HIVE users.

MaxCompute offers many operations for [projects](#), [tables](#), resources, [instances](#), and other objects. You can perform operations on these objects using the console commands and SDK.



Note

- The common commands introduced in this module are mainly targeted at latest version of the [Client](#) .
- If you want to learn how to install and configure clients, see [Install and configure the odpscmd client](#)Quick Start.
- For more information about the SDK, see [MaxCompute SDK introduction](#)MaxCompute SDK introduction.

List of common command

[add](#)、[alias](#)、[alter](#)

[cost](#)、[create](#)

[delete/drop](#)、[desc/describe](#)、[download/get](#)

[extended](#)

[flag/flags](#)、[functions](#)

[get](#)

[help](#)、[history](#)

[jar/mapreduce](#)

[kill](#)

[lifecycle](#)、[list](#)

[odpscmd](#)

[partition](#)

[q/quit](#)

[resource](#)

[set](#)、[show](#)、[status](#)

[tunnel](#)

[upload](#)

[wait](#)、[who](#)

Use limits

When you perform resource operations, please note that the size of each resource file should not exceed 500 MB, and the total size of resources referenced by a single SQL or MapReduce task should not exceed 2048 MB. For more restrictions, see [MR limits](#).

1.2. Project operations

This topic describes how to enter a project, view projects, specify project properties, and configure IP address whitelists for projects.

Enter a project

- Syntax

```
use <project_name>;
```

- Description

- You can run this command to enter a specific project. You must have access permissions on the project. After you enter the project, you can manage all objects in the project.
- If the project that you specify does not exist or you do not have access permissions on the project, an error is returned.

- Examples

- Enter a specific project.

```
-- Enter the my_project_test project. The user has access permissions on this project.  
odps @ my_project>use my_project_test;
```

- Access an object in a specific project.

```
-- Query the test_src table in the my_project_test project.  
odps @ my_project_test>SELECT * FROM test_src;
```

MaxCompute automatically searches for the table in the my_project_test project. If this table exists, data in the table is returned. If this table does not exist, an error is returned.

- Access an object in another project. To achieve this, you must specify the project name.

```
-- Access the test_src table in the my_project2 project from the my_project_test project.  
odps @ my_project_test>select * from my_project2.test_src;
```

Note

- All the preceding commands must be run on the odpscmd client. All command keywords, project names, table names, and column names in MaxCompute are not case-sensitive.
- Each workspace that you create in the DataWorks console is a MaxCompute project.
- You cannot run commands to create or delete projects in MaxCompute. You can manage projects in the DataWorks console. For more information, see [Workspaces](#).

View projects

- Syntax

```
list projects;
```

 **Note** You can run this command only on odpscmd 0.30.2 or later.

- Description

You can run this command by using your Alibaba Cloud account to view the projects that are created by the account.

Specify project properties

- Syntax

```
setproject <KEY>=<VALUE>;
```

- Description

- You can run this command to specify project properties in the `<KEY>=<VALUE>` format for the current project. For example, to enable a full table scan, run the following command:

```
setproject odps.sql.allow.fullscan=true;
```

 **Note** The command that is used to specify project properties takes effect within five minutes. Check the result five minutes after the command succeeds.

- If you do not specify any properties in the `<KEY>=<VALUE>` format in the command, the property settings of the current project are displayed.

```
-- View the project properties that are configured by running the setproject command.  
setproject;
```

The following table describes details about project properties.

Property	Role	Description	Valid value
odps.sql.allow.fullscan	Project Owner	Specifies whether to allow a full table scan.	■ True ■ False
odps.table.drop.ignorenonexistent	All users	Specifies whether to report an error if the table that you want to delete does not exist.	■ True: No error is reported. ■ False: An error is reported.

Property	Role	Description	Valid value
odps.table.lifecycle	Project Owner	- Optional: The lifecycle clause is optional in a table creation statement. If you do not set the lifecycle for a table, the table does not expire. - Mandatory: The lifecycle clause is required in a table creation statement. - Inherit: If you do not set a lifecycle for a table when you create the table, the value of odps.table.lifecycle.value is used as the lifecycle of the table.	- Optional - Mandatory - Inherit
odps.table.lifecycle.value	Project Owner	The lifecycle of the table. Unit: days.	1 to 37231 (default value)
odps.security.ip.whitelist	Project Owner	The whitelist of IP addresses that are authorized to access the project over the classic network.	A list of IP addresses separated with commas (,)
odps.security.vpc.whitelist	Project Owner	The whitelist of IP addresses that are authorized to access the project in a specific virtual private cloud (VPC).	A list of records that consist of region IDs, VPC IDs, and IP addresses. Records are separated with commas (,).
odps.instance.retain.days	Project Owner	The retention period for instance information, in days.	3 to 30
READ_TABLE_MAX_ROW	Project Owner	The maximum number of data entries that can be returned by a SELECT statement.	1 to 10000
odps.sql.type.system.odps2	Project Owner	Specifies whether to enable the MaxCompute V2.0 data type edition.	- True - False

Property	Role	Description	Valid value
odps.sql.hive.compatible	Project Owner	Specifies whether to enable the Hive-compatible data type edition. MaxCompute supports Hive syntax, such as <code>inputRecordReader</code> , <code>outputRecordReader</code> , and <code>serde</code> , only after the Hive-compatible data type edition is enabled.	■ True ■ False

 **Note** Enabling the MaxCompute V2.0 data type edition has the following impacts:

- Some implicit type conversions are disabled. For example, if the data type is converted from `STRING` to `BIGINT`, from `STRING` to `DATETIME`, from `DOUBLE` to `BIGINT`, from `DECIMAL` to `DOUBLE`, or from `DECIMAL` to `BIGINT`, precision may be reduced, or errors may occur. You can use the `CAST` function to forcibly convert the data type.
- The constant type changes. A single integer constant, such as 123, falls into the `BIGINT` type in the original data type edition, but it falls into the `INT` type in the MaxCompute V2.0 data type edition.
- User-defined function (UDF) resolution results may change. Assume that a UDF contains two overloads: `BIGINT` and `INT`. After UDF resolution, the `BIGINT` overload is obtained in the original data type edition, but the `INT` overload may be obtained in the MaxCompute V2.0 data type edition.

Configure IP address whitelists

MaxCompute allows you to configure the whitelists of IP addresses that are authorized to access projects over the classic network or a VPC. The `odps.security.ip.whitelist` parameter specifies the IP address whitelist for the classic network. The `odps.security.vpc.whitelist` parameter specifies the IP address whitelist for a VPC.

• Syntax

- If you configure the IP address whitelist only for the classic network, access requests over the classic network are limited, and access requests over a VPC are prohibited.

Configuration command:

```
setproject odps.security.ip.whitelist=IP Address odps.security.vpc.whitelist=\N;
```

When you configure the IP address whitelist for the classic network, add the IP address of the device on which the MaxCompute client is installed to the whitelist. Otherwise, your access request is denied.

- If you configure the IP address whitelist only for a VPC, access requests over the VPC are limited, and access requests over the classic network are prohibited. Configuration command:

```
setproject odps.security.ip.whitelist=\N odps.security.vpc.whitelist=RegionID_VPCID[IP Address];
```

- If you configure the IP address whitelists for both the classic network and VPC, access requests over the classic network and VPC are limited. Configuration command:

```
setproject odps.security.ip.whitelist=IP Address odps.security.vpc.whitelist=RegionID_VPCID[IP Address];
```

- You can modify the configuration of an IP address whitelist. Configuration commands:

```
-- Modify the configuration of an IP address whitelist for the classic network.
```

```
setproject odps.security.ip.whitelist=IP Address;
```

```
-- Modify the configuration of an IP address whitelist for a VPC.
```

```
setproject odps.security.vpc.whitelist=RegionID_VPCID[IP Address];
```

- You can disable the IP address whitelist feature.

```
setproject odps.security.ip.whitelist= odps.security.vpc.whitelist= ;
```

- Description

- After an IP address whitelist is configured for a project, only IP addresses in the whitelist, such as the outbound IP addresses of the odpscmd client or SDK, can be used to access the project.
- The IP address whitelist takes effect five minutes after it is configured.

 **Note** If you are blocked from a project due to misoperations, to Alibaba Cloud for technical support.

- Parameters

- IP Address: a list of IP addresses. For a VPC, the value is a list of private IP addresses in the VPC.

You can specify IP addresses in the following formats in an IP address whitelist and separate the IP addresses with commas (,):

- IPv4 or IPv6 addresses. Example: 192.168.0.0 or 2001:db8::
- IP addresses with subnet masks. Example: 172.12.0.0/16 or 2001:db8::/32
- CIDR blocks. Example: 192.168.10.0-192.168.255.255 or 2001:db8:1:1:1:1:1-2001:db8:4:4:4:4:4

- **RegionID**: the ID of the region to which the VPC belongs.

Region	Region ID
China (Zhangjiakou-Beijing Winter Olympics)	cn-zhangjiakou
China (Beijing)	cn-beijing
China (Shenzhen)	cn-shenzhen
China (Chengdu)	cn-chengdu
China (Shanghai)	cn-shanghai
China (Hangzhou)	cn-hangzhou
Shanghai Tower	cn
China (Hong Kong)	cn-hongkong
Singapore (Singapore)	ap-southeast-1
Australia (Sydney)	ap-southeast-2
Malaysia (Kuala Lumpur)	ap-southeast-3
Indonesia (Jakarta)	ap-southeast-5
Japan (Tokyo)	ap-northeast-1
Germany (Frankfurt)	eu-central-1
US (Silicon Valley)	us-west-1
US (Virginia)	us-east-1
India (Mumbai)	ap-south-1
UAE (Dubai)	me-east-1
UK (London)	eu-west-1

- VPCID: the ID of the VPC.
 - If this is your first time to configure an IP address whitelist for a VPC, log on to the **MaxCompute client** and run the following command to obtain the VPC ID:

```
whoami;
```

The following information is returned.



 **Note** This command can be used only if the version of the MaxCompute client is V0.31.2 or later.

- If you want to add an IP address to an established whitelist for a VPC, obtain the region ID from the error message returned when you use the IP address to access MaxCompute for the first time. The error message returned because the new IP address is not authorized.



- Examples

For more information, see [Configure an IP address whitelist](#).

1.3. Table-level operations

This topic describes how to perform table-level operations by executing common statements on the MaxCompute client. For example, you can create a table, delete a table, or view information about a table.

You can execute common statements on the MaxCompute client to manage tables. You can also use the visualized table management feature of DataWorks to add tables to favorites, apply for permissions on tables, and view partition information. For more information about the table-level operations in the DataWorks console, see [View table details](#).

Create a table

- Syntax

```
CREATE TABLE [IF NOT EXISTS] table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[LIFECYCLE days]
[AS select_statement]
CREATE TABLE [IF NOT EXISTS] table_name
LIKE existing_table_name
```

- Features

- Tables can be divided into partitioned tables and non-partitioned tables.

- If you execute a CREATE TABLE statement without the `IF NOT EXISTS` clause to create a table that already exists, the system returns an error. If you execute a CREATE TABLE statement with the `IF NOT EXISTS` clause to create a table, the system returns a success message regardless of whether a table with the same name exists. If a table to be created shares the same name with an existing table, the table is not created and the metadata of the existing table is not changed.
- Parameters
 - `table_name`: the name of the table. The table name is not case-sensitive and cannot contain special characters. It can contain only letters, digits, and underscores (_). The name must start with a letter and can be up to 128 bytes in length. Otherwise, an error is returned.
 - `col_name`: the name of the column. The column name is not case-sensitive and cannot contain special characters. It can contain only letters, digits, and underscores (_). The name must start with a letter and can be up to 128 bytes in length. Otherwise, an error is returned.
 - `col_comment`: the comment about the column. The comment must be a valid string and can be up to 1,024 bytes in length. Otherwise, an error is returned.
 - `table_comment`: the comment about the table. The comment must be a valid string and can be up to 1,024 bytes in length. Otherwise, an error is returned.
 - `data_type`: the data type of the column. The following data types are supported: BIGINT, DOUBLE, BOOLEAN, DATETIME, DECIMAL, and STRING. For more information, see [Date types](#).
 - `PARTITIONED BY`: the partition key of the table. In MaxCompute V1.0, partition key columns must be of the STRING type. MaxCompute V2.0 supports partition key columns of the TINYINT, SMALLINT, INT, BIGINT, VARCHAR, and STRING types. If you use a partition key to partition a table, a full table scan is not required when you add partitions, update partition data, or read partition data. This improves processing efficiency.

 Note

- A partition key value cannot contain double-byte characters. It must start with a letter and can contain letters, digits, and some supported special characters. It can be up to 128 bytes in length. Supported special characters are spaces, colons (:), underscores (_), dollar signs (\$), number signs (#), periods (.), exclamation points (!), and at signs (@). The behavior of other characters is not defined, such as escape characters `\t`, `\n`, and `/`.
- A table can contain a maximum of six partition levels. By default, a table can contain a maximum of 60,000 partitions. You can adjust the maximum number of partitions for a table based the specific project.

- **LIFECYCLE**: the lifecycle of the table. Only positive integers are supported. Unit: days.
 - **Non-partitioned tables**: If the data in a non-partitioned table remains unchanged for the number of days specified by days after the last data modification, MaxCompute executes a statement such as DROP TABLE to reclaim the table.
 - **Partitioned tables**: MaxCompute determines whether to reclaim a partition based on its LastDataModifiedTime value. Unlike non-partitioned tables, a partitioned table is not deleted even if all of its partitions are reclaimed. You can configure lifecycles for tables, but not for partitions.

 **Note** If you execute a CREATE TABLE LIKE statement to create a table based on the definition of another table, the lifecycle settings of the source table are not duplicated.

• Examples

```
-- Create a partitioned table named sale_detail if no table with this name exists.
CREATE TABLE IF NOT EXISTS sale_detail(
  shop_name  STRING,
  customer_id STRING,
  total_price DOUBLE)
PARTITIONED BY (sale_date STRING,region STRING);

-- Create a non-partitioned table named test1.
CREATE TABLE test1 (key STRING);

-- Create a partitioned table named test2.
CREATE TABLE test2 (key BIGINT) PARTITIONED BY (pt STRING, ds STRING);

-- Create a table named test3 with a lifecycle.
CREATE TABLE test3 (key BOOLEAN) PARTITIONED BY (pt STRING, ds STRING) LIFECYCLE 100;

-- Create a table named test4 based on the definition of the test3 table, without replicating the lifecycle settings.
CREATE TABLE test4 like test3;

-- Create a table named test5 based on the data of the test2 table, without replicating the partition and lifecycle settings. This operation copies only the data of the test2 table to the test5 table. If the test2 table has no data, the test5 table is an empty table. Data import will be described later in this document.
CREATE TABLE test5 as select * from test2;
```

Note

- For more information about how to add a partition, see [Partition and column operations](#).
- For more information about how to modify the lifecycle, see [Lifecycle management operations](#).
- For more information about table-level operations, see [Table-level operations](#).

Change the owner of a table

- Syntax

```
ALTER TABLE table_name CHANGEOWNER to new_owner;
```

- Features

This statement is used to change the owner of a table.

- Examples

```
-- Change the owner of the test1 table to ALIYUN$xxx@aliyun.com.  
ALTER TABLE test1 CHANGEOWNER to 'ALIYUN$xxx@aliyun.com';
```

Delete a table

- Syntax

```
DROP TABLE [IF EXISTS] table_name;
```

- Features

- This statement is used to delete a table.
- If you execute a DROP TABLE statement without the `IF EXISTS` clause to delete a table that does not exist, the system returns an error. If you execute a DROP TABLE statement with the `IF EXISTS` clause to delete a table, the system returns a success message regardless of whether the table exists.

- Examples

```
-- Delete the sale_detail table. If the table exists, the system returns a success message.  
DROP TABLE sale_detail;  
  
-- Delete the sale_detail table. The system returns a success message regardless of whether the sale_detail table exists.  
DROP TABLE IF EXISTS sale_detail;
```

View table information

- Syntax

- View information about a table or view.

```
DESC <table_name>;
```

table_name: the name of the table or view.

- View information about a foreign table.

```
DESC extended <table_name>;
```

table_name: the name of the foreign table.

- Features

This statement returns the following information about a table:

- **Owner:** the owner of the table.
- **Project:** the project to which the table belongs.
- **CreateTime:** the time when the table was created.
- **LastDDLTime:** the last time when a DDL operation was performed.
- **LastModifiedTime:** the last time when the table was modified.
- **InternalTable:** indicates that the described object is a table. The value is always YES.
- **Size:** the storage capacity occupied by compressed table data, which is generally one fifth the size of the original data. Unit: bytes.

 **Note** The size of table data in the execution results of `DESC table_name` includes the size of the data in the recycle bin. To view the size of table data except for the data in the recycle bin, execute the `PURGE TABLE table_name;` statement before you execute the `DESC table_name` statement. To view the details of the data in the recycle bin, execute the `SHOW RECYCLEBIN;` statement.

- **Native Columns:** the information about non-partition key columns, including the column name, type, and description.
- **Partition Columns:** the information about partition key columns, including the column name, type, and description.
- **Extended Info:** the information about foreign tables, including StorageHandler and Location.

 **Note**

- For a non-partitioned table, the Partition Columns section is not displayed.
- For a view, the InternalTable parameter is replaced by the VirtualView parameter, whose value is always YES. The Size parameter is replaced by the ViewText parameter, which defines the view, such as `SELECT * FROM src`. For more information about views, see [View-related operations](#).

- **Examples**

```
-- Query information about the partitioned table sale_detail.
odps@ project_name>DESC sale_detail;
+-----+
| Owner: ALIYUN$odpsuser@aliyun.com | Project: test_project |
| TableComment: |
+-----+
| CreateTime:      2014-01-01 17:32:13 |
| LastDDLTime:     2014-01-01 17:57:38 |
| LastModifiedTime: 1970-01-01 08:00:00 |
+-----+
| InternalTable: YES | Size: 0 |
+-----+
| Native Columns: |
+-----+
| Field | Type | Comment |
+-----+
| shop_name | string | |
| customer_id | string | |
| total_price | double | |
+-----+
| Partition Columns: |
+-----+
| sale_date | string | |
| region | string | |
+-----+
```

View partition information

- Syntax

```
DESC table_name PARTITION(pt_spec);
```

- Features

This statement is used to query the partition information about a partitioned table.

- Examples

```
odps@ project_name>DESC meta.m_security_users PARTITION(ds='20151010');
+-----+
| PartitionSize: 2109112 |
+-----+
| CreateTime:      2015-10-10 08:48:48 |
| LastDDLTime:     2015-10-10 08:48:48 |
| LastModifiedTime: 2015-10-11 01:33:35 |
+-----+
OK
```

List all tables in a project

- Syntax

```
SHOW TABLES;
SHOW TABLES LIKE 'chart';
```

- Features

- `SHOW TABLES` : shows all tables in the current project.
- `SHOW TABLES LIKE 'chart'` : shows the tables whose names match `chart` in the current project. You can use regular expressions in the statement to filter tables.

- Examples

```
odps@ project_name>SHOW TABLES;
odps@ project_name>SHOW TABLES LIKE 'ods_brand*';
ALIYUN$odps_user@aliyun.com:table_name
.....
```

- `ALIYUN`: indicates that the table was created by an Alibaba Cloud user.
- `odps_user@aliyun.com` : the name of the user who created the table.
- `table_name` : the name of the table.

List all partitions

- Syntax

```
SHOW PARTITIONS table_name;
```

- Features

This statement is used to list all partitions of a table. An error is returned if the table does not exist or the table is not a partitioned table.

- Examples

```
odps@ project_name>SHOW PARTITIONS table_name;
partition_col1=col1_value1/partition_col2=col2_value1
partition_col1=col1_value2/partition_col2=col2_value2
.....
```

- `partition_col1` and `partition_col2` indicate the partition key columns of the table.
- `col1_value1` , `col2_value1` , `col1_value2` , and `col2_value2` indicate the values in the matching columns.

Add columns

Syntax

- Add columns to a table

```
ALTER TABLE table_name ADD COLUMNS (col_name1 type1,col_name2 type2...) ;
```

- Add columns and comments at the same time

```
ALTER TABLE table_name ADD COLUMNS (col_name1 type1 comment 'XXX',col_name2 type2 comment '
XXX');
```

Parameters

- `table_name`: the name of the destination table to which you want to add a column. You cannot specify the order of a new column in the table. The new column is added as the last column by default.
- `col_name`, `type` ,`comment`: the name, data type, and comment of the column you want to add.

1.4. Instance operations

This topic describes how to use commands to manage instances on the MaxCompute client.

instance information instance status stop an instance instance task running log

Show Instances/Show P

Syntax

```
SHOW INSTANCES [FROM startdate TO enddate] [number];
SHOW P [FROM startdate TO enddate] [number];
SHOW INSTANCES [-all];
SHOW P [-all];
SHOW P -p <project name>;
```

Description

Displays information about the instances created by the current user. The information in the query result includes `StartTime` (accurate to seconds), `RunTime (s)`, and `Status` (instance status). An instance can be in any of the following states:

- **Running:** The instance is running.
- **Success:** Instance operation is complete.
- **Waiting:** The instance is waiting to run.
- **Failed:** The job failed, but data in the target table is not modified.
- **Suspended:** The instance is suspended.
- **Cancelled:** The instance is stopped.

Parameters

- **startdate and enddate:** returns the instance information during the specified period (from startdate to enddate in the yyyy-mm-dd format. The parameters are optional. If they are not specified, information about instances submitted within three days is returned by default.
- **number:** specifies the number of instances to return. Information about N (specified by number) instances submitted at the time nearest to the current time is returned in chronological order. If it is not specified, information about all instances that meet the requirements is returned.
- **-all:** indicates that all instances executed in the current project are returned. A maximum of 50 records can be returned by default. To run this command, you must have the LIST permission on the project. You can use -limit number to show more records. For example, run `show p -all -limit 100` to show 100 instance records in the current project.
- **project name:** indicates the name of a project. To use this parameter, your account must be a member of the project.

InstanceID and the related SQL statement are as follows:

```
odps@ $project_name>show p;
StartTime      RunTime Status InstanceID          Owner      Query
2015-04-28 13:57:55 1s    Success 20150428xxxxxxxxxxxxxxxx ALIYUN$xxxxx@aliyun-inner.com
select * from tab_pack_priv limit 20;
...  ...  ...  ...  ...  ...
...  ...  ...  ...  ...  ...
```

Status Instance

Syntax

```
status instance_id;
```

Description

- Queries the status of a specified instance, such as Success, Failed, Running, or Cancelled.
- If the specified instance is not created by the current user, an exception is returned.

Parameters

instance_id: specifies the instance whose status you want to query. This parameter uniquely identifies an instance.

Example

```
-- View the status of the instance with ID of 20131225123xxxxxxxxxxxxxxxxx.  
odps@ $project_name>status 20131225123xxxxxxxxxxxxxxxxx;  
Success
```

Top Instance

Only a project owner or the administrator is allowed to run this command.

Syntax

```
top instance;  
top instance -all;
```

Description

- top instance** : displays information about the running job submitted by the current account in the current project. The information includes InstanceID, Owner, Type, StartTime, Progress, Status, Priority, RuntimeUsage (CPU/MEM), TotalUsage (CPU/MEM), and QueueingInfo (POS/LEN).
- top instance -all** : displays all jobs that are running in the current project. A maximum of 50 records can be displayed by default. You can use **-limit number** to show more records.

Example

```
odps@ $project_name>top instance;
```

Kill Instance

Syntax

```
kill instance_id;
```

Description

Stops the specified instance. The instance must be in the Running state.

 **Note** The execution of this command is an asynchronous process, which means that the distributed task may still be running even after the system runs this command and returns the result. You can run the **status** command to check the instance status.

Parameters

instance_id: uniquely identifies an instance. It must be the ID of a running instance. If the instance is not running, an error is returned.

Example

```
-- Stop the instance with ID of 20131225123xxxxxxxxxxxxxxxxx.  
odps@ $project_name>kill 20131225123xxxxxxxxxxxxxxxxx;
```

Desc Instance

Syntax

```
desc instance instance_id ;
```

Description

Obtains job information based on a specific instance ID. The information includes SQL, Owner, StartTime, EndTime, and Status.

Parameters

instance_id: uniquely identifies an instance.

Example

```
-- Query job information about the instance with ID of 20150715xxxxxxxxxxxxxxxxx.
odps@ $project_name> desc instance 20150715xxxxxxxxxxxxxxxxx;
ID                20150715xxxxxxxxxxxxxxxxx
Owner              ALIYUN$XXXXXX@alibaba-inc.com
StartTime          2015-07-15 18:34:41
EndTime            2015-07-15 18:34:42
Status             Terminated
console_select_query_task_1436956481295 Success
Query              select * from mj_test;
```

Wait Instance

Syntax

```
wait instance_id;
```

Description

Obtains task running log information based on the specific instance ID. The information includes the Logview link. You can view detailed logs of a task on the Logview.

 **Note** You can only open the Logview for the instances created in the last three days. For the instances created more than three days, you can obtain the Logview links from their logs.

Parameters

instance_id: uniquely identifies an instance.

Example

```
-- View the task running log of the instance with ID of 20170925161122379gxxxxxx.
wait 20170925161122379gxxxxxx;
ID = 20170925161122379g3xxxxxx
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.com/api&p=aliam&i=2017092516
11223xxxxxxdqp&token=XXXXXXvbi6IjEifQ==
Job Queueing...
Summary:
resource cost: cpu 0.05 Core * Min, memory 0.05 GB * Min
inputs:
    alian.bank_data: 41187 (588232 bytes)
outputs:
    alian.result_table: 8 (640 bytes)
Job run time: 2.000
Job run mode: service job
Job run engine: execution engine
M1:
    instance count: 1
    run time: 1.000
    instance time:
        min: 1.000, max: 1.000, avg: 1.000
    input records:
        TableScan_REL5213301: 41187 (min: 41187, max: 41187, avg: 41187
    )
    output records:
        StreamLineWrite_REL5213305: 8 (min: 8, max: 8, avg: 8)
R2_1:
    instance count: 1
    run time: 2.000
    instance time:
        min: 2.000, max: 2.000, avg: 2.000
    input records:
        StreamLineRead_REL5213306: 8 (min: 8, max: 8, avg: 8)
    output records:
        TableSink_REL5213309: 8 (min: 8, max: 8, avg: 8)
```

1.5. Resource operations

This topic describes how to use common commands to manage resources on the MaxCompute client.

add a resource delete a resource view the resource list download resources
You can also search and upload resources by using the visualized online data development tools offered in DataWorks.

Add a resource

Syntax

```
add file <local_file> [as alias] [comment 'cmt'][-f];
add archive <local_file> [as alias] [comment 'cmt'][-f];
add table <table_name> [partition (spec)] [as alias] [comment 'cmt'][-f];
add <PY | JAR> <localfile[.py|.jar]> [COMMENT 'cmt'][-f];
```

Description

Adds a resource to MaxCompute.

Parameters

- file/archive/table/jar/py: indicates the resource type. For more information, see [Resources](#).
- Local_file: indicates the path of the local file. The file name is used as the resource name, which uniquely identifies a resource.
- table_name: indicates the table name in MaxCompute. Currently, external tables cannot be added as resources.
- [PARTITION (spec)]: If the resource you want to add is a partitioned table, MaxCompute considers only a partition as a resource, not the entire partitioned table.
- alias: specifies a resource name. If this parameter is not specified, the file name is used as a resource name by default. Jar and Python resources do not support this parameter.
- [comment 'cmt']: adds a comment for a resource.
- [-f]: overwrites the original resource that has the same name as the current resource. If this parameter is not specified and a duplicate resource name exists, the operation fails.

Example

```
-- Add a partitioned table resource named sale.res to MaxCompute.
odps@ odps_public_dev>add table sale_detail partition (ds='20150602') as sale.res comment 'sale detail on 20150602' -f;
OK: Resource 'sale.res' have been updated.
```

 **Note** Currently, the size of each resource file cannot exceed 500 MB. The resource size referenced by a single SQL or MapReduce task cannot exceed 2,048 MB. For more limits, see [MapReduce使用限制](#).

View the resource list

Syntax

```
LIST RESOURCES;
```

Description

Views all resources in the current project.

Example

```
odps@ $project_name>list resources;
```

Resource Name	Comment	Last Modified Time	Type
1234.txt	2014-02-27 07:07:56	file	
mapred.jar	2014-02-27 07:07:57	jar	

Download resources

Syntax

```
GET RESOURCE <resource_name> <path>;
```

Description

Downloads MaxCompute resources to your local device. The resource type must be FILE, JAR, ARCHIVE, or PY.

Parameters

- **resource_name**: indicates the name of the resource to be downloaded.
- **path**: specifies the path for saving a resource to your local device.

Example

```
odps@ $project_name>get resource odps-udf-examples.jar d:\;
OK
```

Delete a resource

Syntax

```
DROP RESOURCE <resource_name>;
```

Description

Deletes an existing resource.

Parameters

resource_name: indicates the name of the resource you want to delete. That is the resource name specified when you create the resource.

1.6. Functions operations

This article explains how to use common commands to operate functions in the MaxCompute client.

You can also operate functions using the visualized online data development tools in DataWorks.

Create a Function

Command format:

```
CREATE FUNCTION <function_name> AS <package_to_class> USING <resource_list>;
```

Parameters

- **function_name**: An UDF name referenced in SQL.
- **package_to_class**: For Java UDF, this name is a fully qualified class name (from top-level package name to UDF class name). This parameter must be in double quotation marks. And, for Python UDF, this name is a python script name. classname. For both Java UDF and python script, use double quotation (" ") marks to indicate this parameter. And for the name, use quotation marks.
- **resource_list**: Provides resource list used by UDF.
 - Resources that contain UDF code must be included in the list.
 - If the code reads the resource file by the distributed cache interface, this list also contains the list of resource files read by the UDF.
 - The resource list is composed of multiple resource names, separated by a comma (,). The resource list must be in double quotation ("") marks.
 - Specify the project in which the resource is located as follows: <project_name>/resources/<resource_name> .

Example:

- Suppose a Java UDF class org.alidata.odps.udf.examples.Lower is in my_lower.jar, create function my_lower as follows:

```
CREATE FUNCTION my_lower AS 'org.alidata.odps.udf.examples.Lower' USING 'my_lower.jar';
```

- Suppose a Python UDF MyLower is used in project pyudf_test.py, create function my_lower as follows:

```
create function my_lower as 'pyudf_test.MyLower' using 'test_project/resources/pyudf_test.py';
```

- Suppose a Java UDF class com.aliyun.odps.examples.udf.UDTFResource is in udtfexample1.jar, and it depends on file resource file_resource.txt and table resource table_resource1, create function test_udtf as follows:

```
create function test_udtf as 'com.aliyun.odps.examples.udf.UDTFResource' using 'udtfexample1.jar, file_resource.txt, table_resource1, test_archive.zip';
```

 Note

- Similar to the resource files, the UDF duplicate name can be registered only once.
- Generally UDF cannot overwrite system built-in functions. Only the project owner has right to overwrite the built-in functions. If you are using a UDF which overwrites the built-in function, the warning is triggered in Summary after SQL execution.

Drop a Function

Command format:

```
DROP FUNCTION <function_name>;
```

Example:

```
DROP FUNCTION test_lower;
```

List Functions

Command format:

```
list functions; --View all user-defined functions in current project.  
list functions -p my_project; --View all user-defined functions in the project 'my_project'.
```

1.7. SET operations

This topic describes how to use the SET command to set MaxCompute or user-defined system variables.

SET

Syntax

```
set <KEY>=<VALUE>
```

Description: This command is used to set MaxCompute system variables.

 Note

- The `SET` command only supports lower-case letters.
- For project-level settings, see [SetProject](#).

Parameters

- **KEY:** the attribute name that you want to change.
- **VALUE:** the attribute value. The following table describes the attributes and their values.

Attribute	Description	Value range
odps.sql.allow.fullscan	Specifies whether to allow a full scan on a partitioned table.	Valid values: True (allow) and False (prohibit).
odps.stage.map.per.mem	Sets the memory size of each Map worker.	Unit: MB. Default value: 1024 MB.
odps.stage.reducer.mem	Sets the memory size of each Reduce worker.	Unit: MB. Default value: 1024 MB.
odps.stage.joiner.mem	Sets the memory size of each Join worker.	Unit: MB. Default value: 1024 MB.
odps.stage.mem	Sets the total memory size of all workers in a specific MaxCompute job. This attribute has a lower priority than the preceding three attributes.	Unit: MB. No default value.
odps.stage.map.per.split.size	Modifies the input data volume of each Map worker, which is the split size of the input file. You can use this attribute to indirectly control the number of workers in each Map stage.	Unit: MB. Default value: 256 MB.
odps.stage.reducer.num	Modifies the number of workers in each Reduce stage.	No default value.
odps.stage.joiner.num	Modifies the number of workers in each Join stage.	No default value.

Attribute	Description	Value range
odps.stage.num	Modifies the worker concurrency at all stages in a specific MaxCompute job. This attribute has a lower priority than the preceding three attributes.	No default value.
odps.sql.reshuffle.dynamicpt	Sets dynamic partitions to avoid the excessive generation of small files.	Valid values: True and False. Default value: True.  Note If a small number of dynamic partitions are generated, we recommend that you set this parameter to False to avoid data skew.
odps.sql.type.system.odps2	Specifies whether to use new data types. You need to set it to True if new data types , such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, and BINARY, are used in SQL operations. This setting is required for sessions.	Valid values: True (enabled) and False (disabled). Default value: False.

Attribute	Description	Value range
odps.sql.hive.compatible	Specifies whether to enable the Hive compatibility mode. MaxCompute supports various syntax specified by Hive, such as inputRecordReader, outputRecordReader, and Serde only after the Hive compatibility mode is enabled. This setting is required for sessions.	Valid values: True (enabled) and False (disabled). Default value: False.
odps.sql.executionengine.coldata.deep.buffer.size	Adjusts the cache size requested for storing a column that contains	○ Scenarios ■ The output table contains too many complex data types. ■ The size of a variable of a complex data type in the output table is too large.

ze.max Attribute	complex data to Description. improve write performance.	○ Instructions Value range ■ The default value is 67108864 bytes (64 MB). Unit: bytes. ■ By default, MaxCompute reserves 192 MB memory (64 × 3) for table write operations if the schemas of three columns in the output table are of complex data types, such as STRING, MAP, STRUCT, ARRAY, and BINARY. The cache requested for each column is used to store data in <code>batch row count</code> that corresponds to this column. ■ If you know in advance that variables of complex data types in the table consume less space, we recommend that you set this parameter to a smaller value. For example, if the size of each variable of a complex data type does not exceed 1024 bytes and the default value (1024) of <code>batch row count</code> is used, you can set the value to 1048576 (1024 × 1024). <pre>set odps.sql.executionengine.coldata.deep.buffer.size.max=1048576;</pre> ■ If you know in advance that the size of each variable of a complex data type is between 7 MB and 8 MB, and <code>batch row count</code> is set to 32, you can adjust this parameter to a value in bytes that equals to 256 MB (8 × 32). ■ If the output of a task or the small table in the <code>MAPJOIN</code> operation contains complex data types, then

Examples

- Adjust the cache size requested for storing a column that contains complex data to improve write performance.

```
set odps.sql.executionengine.coldata.deep.buffer.size.max=1048576;
```

- Set the size of data read by each Mapper to 256 MB.

```
set odps.stage.mapper.split.size=256
```

Show Flags

Syntax

```
show flags;
```

Description: This command is used to display the parameters configured by using the SET command.

 **Note** This command takes effect only at the project level. For more information about how to set project-level flags, see [Project operations](#).

1.8. Other operations

Alias command

The ALIAS command reads different resources (data) using a fixed resource name in **MapReduce** or **UDF** without modifying the code.

Command format:

```
ALIAS <alias>=<real>;
```

Action:

Create alias for a resource.

Example:

```
ADD TABLE src_part PARTITION (ds='20121208') AS res_20121208;
ADD TABLE src_part PARTITION (ds='20121209') AS res_20121209;
ALIAS resName=res_20121208;
jar -resources resName -libjars work.jar -classpath ./work.jar com.company.MainClass args ... ;//job 1
ALIAS resName=res_20121209;
jar -resources resName -libjars work.jar -classpath ./work.jar com.company.MainClass args ... ;//job 2
```

In the preceding example, resource alias **resName** refers to different resource tables in two jobs. Different data can be read without modifying the code.

Cost SQL

Command format:

```
cost sql <SQL Sentence>;
```

Action:

Estimate an SQL measurement message, including the size of the input data, the number of UDFs, and the SQL complexity level.

 **Note** Use the following information for reference purpose only. Refrain from using it as an actual charging standard.

Example:

```
odps@ $odps_project >cost sql select distinct project_name, user_name from meta.m_security_users distribute by project_name sort by project_name;
```

```
ID = 20150715113033121xxxxxxxxx
```

```
Input:65727592 Bytes
```

```
UDF:0
```

```
Complexity:1.0
```

2.Data upload and download

2.1. Overview

MaxCompute data import and export

This topic describes how to upload data to or download data from MaxCompute, including service connection, SDKs, tools, data import and export, and cloud migration.

data import and export DataHub Tunnel

MaxCompute provides two data upload and download channels:

- DataHub: provides real-time data upload and download services. It includes the OGG, Flume, Logstash, and Fluentd plug-ins.
- Tunnel: provides batch data upload and download services. It includes the MaxCompute client, DataWorks, DTS, Sqoop, Kettle plug-in, and MaxCompute Migration Assist (MMA).

DataHub and Tunnel provide their own SDKs. The data upload and download tools derived from these SDKs allow you to upload or download data in a variety of scenarios. For more information, see [MaxCompute Tunnel overview](#).

The preceding tools provide sufficient capabilities for most scenarios to migrate data to the cloud. The subsequent topics detail the tools and describe cloud migration scenarios, such as [data migration from Hadoop to MaxCompute](#), database synchronization, and log collection, helping you select proper technical schemes.

 **Note** For offline data synchronization, we recommend that you use the data integration feature.

Limits

- Limits of data uploads by using Tunnel:
 - You cannot run Tunnel commands to upload or download data of the ARRAY, MAP, and STRUCT types.
 - There is no limit on the upload speed. The upload speed depends on the network bandwidth and server performance.
 - There is a limit on the number of retries. When the number of retries exceeds the limit, the next block is uploaded. After data is uploaded, you can execute the `select count(*) from table_name` statement to check whether any data is lost.
 - A project supports a maximum of 2,000 concurrent Tunnel connections by default.
 - On the server, the lifecycle for each session spans 24 hours after it is created. A session can be shared among processes and threads on the server, but you must make sure that each block ID is unique.
- Limits of data uploads by using DataHub:
 - The size of each field cannot exceed its upper limit. For more information, see [Date types](#).

 **Note** The size of a STRING-type field cannot exceed 8 MB.

- During the upload, multiple data entries are packaged.

- Limits of TableTunnel SDK interfaces:
 - The value of a block ID must be greater than or equal to 0 and less than 20000. The volume of data you want to upload in a block cannot exceed 100 GB.
 - The lifecycle of a session is 24 hours. If you want to transfer large volumes of data, we recommend that you transfer your data in multiple sessions.
 - The lifecycle of an HTTP request that corresponds to a RecordWriter is 120 seconds. If no data flows over an HTTP connection within 120 seconds, the server closes the connection.

2.2. Connection to data tunnel service

In different network environments, you need to choose different service addresses (Endpoints) to connect services, otherwise you will not be able to initiate requests to services.

Both DataHub and Tunnel use different endpoints in different network environments. Depending on the network environment, select the appropriate service address or endpoint, to connect to the service. Select the appropriate address or endpoint for your network to be able to send requests to the service.



Note

Different network connections may affect your **Billing**.

For detailed endpoints information for different network environments, see **Endpoints and Data Centers** Access Domains and Data Centers.

2.3. Selection of tools to migrate data to the cloud

MaxCompute provides a variety of data upload and download tools, which can be used in different scenarios to migrate data to the cloud. This topic describes the selection of data transmission tools in three typical scenarios.

Hadoop data migration synchronization of data in a database log collection

Hadoop data migration

You can use MaxCompute Migration Assist (MMA), Sqoop, and DataWorks to migrate Hadoop data.

- If you use DataWorks, DataX is required.
- If you use Sqoop, a MapReduce job is executed on the original Hadoop cluster for distributed data transmission to MaxCompute. For more information, visit **Apache Sqoop**.

Synchronization of data in a database

To synchronize data from a database to MaxCompute, you must select a tool based on the database type and synchronization policy.

- Use DataWorks for offline batch synchronization. DataWorks supports a wide range of database types, which include MySQL, SQL Server, and PostgreSQL. You can also perform instance-related operations based on **Create a batch synchronization node**.

Log collection

You can use tools such as Flume, Fluentd, and Logstash to collect logs.

2.4. Introduction to the tools

MaxCompute supports a wide range of data upload and download tools. The source code for most of the tools can be found and maintained on the open source community GitHub. You can select the appropriate tools to upload and download data based on application scenarios.

Data Integration of DataWorks upload and download MaxCompute client DTS (Tunnel)

Alibaba Cloud DTplus products

- MaxCompute client (Tunnel)
 - Based on the [batch data tunnel](#) SDK, the client provides built-in Tunnel commands for data uploads and downloads. For more information about Tunnel commands, see [Tunnel commands](#).
 - For information about client installation and usage, see [Client](#).

 **Note** This is an open source project which you can view by visiting [aliyun-odps-console](#).

- Data Integration of DataWorks (Tunnel)

Data Integration of DataWorks is a stable, efficient, and scalable data synchronization platform provided by Alibaba Cloud. It is designed to provide full offline and incremental real-time data synchronization, integration, and exchange services for the heterogeneous data storage systems on Alibaba Cloud.

Data synchronization tasks support the following data sources: MaxCompute, ApsaraDB for RDS (MySQL, SQL Server, and PostgreSQL), Oracle, FTP, AnalyticDB (ADS), OSS, ApsaraDB for Memcache, and DRDS.

- DTS (Tunnel)

[Data Transmission Service \(DTS\)](#) is an Alibaba Cloud data service that supports data exchange among multiple data sources, such as Relational Database Management System (RDBMS), NoSQL, and Online Analytical Processing (OLAP). It provides data transmission features, such as data migration, real-time data subscription, and real-time data synchronization.

DTS supports data synchronization from ApsaraDB for RDS and MySQL instances to MaxCompute tables. Other data sources are not supported.

Open source products

- Sqoop (Tunnel)

Sqoop 1.4.6 in the community is further developed to provide enhanced MaxCompute support. It imports data from relational databases such as MySQL and data from HDFS or Hive to MaxCompute tables. It also exports data from MaxCompute tables to relational databases such as MySQL.

 **Note** This is an open source project which you can view by visiting [aliyun-maxcompute-data-collectors](#).

- Kettle (Tunnel)

Kettle is an open source extract, transform, load (ETL) tool that is developed in Java. It runs on Windows, Unix, or Linux, and provides graphic interfaces for you to define the data transmission topology by using drag-and-drop components.

 **Note** This is an open source project which you can view by visiting [aliyun-maxcompute-data-collectors](#).

- Apache Flume (DataHub)

Apache Flume is a distributed and reliable system. It efficiently collects large volumes of log data from different data sources and then aggregates and stores the data in a centralized data storage. It supports multiple Source and Sink plug-ins.

The DataHub Sink plug-in of Apache Flume allows you to upload log data to DataHub in real time and archive the data in MaxCompute tables.

 **Note** This is an open source project which you can view by visiting [aliyun-maxcompute-data-collectors](#).

- Fluentd (DataHub)

Fluentd is an open source software product. It collects logs, such as application logs, system logs, and access logs, from various sources. It allows you to use plug-ins to filter log data and store the data in different data processors, including MySQL, Oracle, MongoDB, Hadoop, and Treasure Data.

The DataHub plug-in of Fluentd allows you to upload log data to DataHub in real time and archive the data in MaxCompute tables.

- Logstash (DataHub)

Logstash is an open source log collection and processing framework. The logstash-output-datahub plug-in allows you to import data to DataHub. This tool can be easily configured to collect and transmit data. It can be used together with MaxCompute or StreamCompute to easily create an all-in-one streaming data solution from data collection to analytics.

The DataHub plug-in of Logstash allows you to upload log data to DataHub in real time and archive the data in MaxCompute tables.

- OGG (DataHub)

The DataHub plug-in of OGG allows you to incrementally synchronize data in an Oracle database to DataHub in real time and archive the data in MaxCompute tables.

 **Note** This is an open source project which you can view by visiting [aliyun-maxcompute-data-collectors](#).

2.5. Tunnel SDK

2.5.1. MaxCompute Tunnel overview

This topic provides an overview of MaxCompute Tunnel, which is a data tunnel that is used for data uploads and downloads.

Tunnel SDK data download data upload

Tools for data uploads and downloads provided by MaxCompute are based on the Tunnel SDK. The following table describes the interfaces of the Tunnel SDK, which may differ based on the SDK versions. For more information, visit [SDK Java Doc](#).

Interface	Description
TableTunnel	The entry-class interface that is used to access MaxCompute Tunnel. You can access MaxCompute and MaxCompute Tunnel over the Internet or an internal network on Alibaba Cloud. Tunnel-based data downloads over an internal network are free of charge.
TableTunnel.UploadSession	The session that uploads data to a MaxCompute table.
TableTunnel.DownloadSession	The session that downloads data from a MaxCompute table.
InstanceTunnel	The entry-class interface that is used to access MaxCompute Tunnel. You can access MaxCompute and MaxCompute Tunnel over the Internet or an internal network on Alibaba Cloud. Tunnel-based data downloads over an internal network are free of charge.
InstanceTunnel.DownloadSession	The session that downloads data to a MaxCompute SQL instance. The SQL instance must start with the SELECT keyword and is used to query data.

Note

- If you use Maven, search for odps-sdk-core in the [Maven database](#) to find different versions of the SDK for Java. The configuration is as follows:

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-core</artifactId>
  <version>0.24.0-public</version>
</dependency>
```

- For more information about the SDK, visit [SDK Java Doc](#).
- For more information about service connections, see [Configure endpoints](#).

2.5.2. TableTunnel

This topic provides a definition and describes the processes and limits of TableTunnel, which itself provides an ingress class for you to use the MaxCompute Tunnel service to upload and download tables.

Definition

The TableTunnel interface is defined as follows:

```
public class TableTunnel {  
    public DownloadSession createDownloadSession(String projectName, String tableName);  
    public DownloadSession createDownloadSession(String projectName, String tableName, PartitionSpec  
partitionSpec);  
    public UploadSession createUploadSession(String projectName, String tableName);  
    public UploadSession createUploadSession(String projectName, String tableName, PartitionSpec parti  
tionSpec);  
    public DownloadSession getDownloadSession(String projectName, String tableName, PartitionSpec pa  
rtitionSpec, String id);  
    public DownloadSession getDownloadSession(String projectName, String tableName, String id);  
    public UploadSession getUploadSession(String projectName, String tableName, PartitionSpec partiti  
onSpec, String id);  
    public UploadSession getUploadSession(String projectName, String tableName, String id);  
}
```

For more information, see [Java-sdk-doc](#).

Parameter description:

- **lifecycle**: the period of time that starts when a TableTunnel instance is created and ends when the service process is completed.
- **UploadSession and DownloadSession**: the objects that you can create by using TableTunnel. After creating an UploadSession or DownloadSession object, you can call a session, TableTunnel.UploadSession or TableTunnel.DownloadSession, to upload or download data.
- **Session**: the period of time in which a table or partition is uploaded or downloaded. A session consists of one or more HTTP requests to the Tunnel RESTful API actions that you call.
- **TableTunnel.UploadSession**: This session offers the same capabilities as the `INSERT INTO` statement. You can create multiple sessions to upload the same table or partition, and these sessions do not affect each another. The data uploaded by each session is stored to a unique directory.
- **RecordWriter**: In TableTunnel.UploadSession, each RecordWriter class corresponds to an HTTP request and is uniquely identified by its block ID. The block ID is the name of the file corresponding to the RecordWriter class.
- **blockId**: If you use the same block ID to open a RecordWriter class multiple times in the same session, the data uploaded by the RecordWriter class that is the last to call the `close()` function overwrites all previous data. This is helpful in retransmitting data of a block in case

that the block data fails to be transmitted.

Process

1. The `RecordWriter.write()` function uploads your data as files to a temporary directory.
2. The `RecordWriter.close()` function moves the files from the temporary directory to the *Data* directory.
3. The `session.commit()` function moves each file in the *Data* directory to the directory where the corresponding table is located, and updates the table metadata accordingly. In this way, data moved into a table by the current task can be visible to the other MaxCompute tasks such as SQL and MapReduce.

Limits

- The number used for the block ID must be greater than or equal to 0 and less than 20000. The size of data to be uploaded in a block cannot exceed 100 GB.
- A session is uniquely identified by its ID. The lifecycle of a session is 24 hours. If your session times out due to large data uploads, you will need to split your data into multiple sessions to mitigate the chance of session timeouts.
- The lifecycle of an HTTP request of a `RecordWriter` class is 120s. If there is no inbound traffic over an HTTP connection within 120 seconds, the server closes the connection.

 **Note** HTTP provides an 8-KB buffer. When you call the `RecordWriter.write()` function, your data can be saved to the buffer with no inbound traffic over the corresponding HTTP connection. In such a case, you can call the `TunnelRecordWriter.flush()` function to make this data stored in the buffer into inbound traffic over the HTTP connection.

- When you create `RecordWriter` classes to write logs to MaxCompute, the `RecordWriter` classes are likely to time out due to unexpected traffic fluctuations. Therefore, we recommend that you:
 - Do not create a `RecordWriter` class for each of your data records because each `RecordWriter` class corresponds to a file. If you create a `RecordWriter` class for every data record, a large number of small files are produced, degrading the overall performance of MaxCompute.
 - Do not create a `RecordWriter` class to write data in batches until the size of cached code reaches 65 MB.
- The lifecycle of a `RecordReader` class is 300s.

2.5.3. InstanceTunnel

This topic describes how to use `InstanceTunnel` to download the execution results of a `SELECT` statement.

Description

The following code describes `InstanceTunnel`. For more information, visit [Java-sdk-doc](#).

```
public class InstanceTunnel{
    public DownloadSession createDownloadSession(String projectName, String instanceID);
    public DownloadSession createDownloadSession(String projectName, String instanceID, boolean limit
    Enabled);
    public DownloadSession getDownloadSession(String projectName, String id);
}
```

Parameters:

- projectName: the name of a project
- instanceID: the ID of an instance

Limits

Although InstanceTunnel provides an easy way to obtain instance execution results, it is subject to the following permission limits to ensure data security:

- If the number of records does not exceed 10,000, all users who have read permissions on the specific instance can use InstanceTunnel to download the data. This is consistent with data query by calling a RESTful API.
- If the number of records exceeds 10,000, only users who have read permissions on all the source tables in the SQL query statement can use InstanceTunnel to download the results.

2.5.4. UploadSession

This paper introduces the UploadSession interface.

UploadSession interface defination

The UploadSession interface is defined as follows:

```
public class UploadSession {
    UploadSession(Configuration conf, String projectName, String tableName,
        String partitionSpec) throws TunnelException;
    UploadSession(Configuration conf, String projectName, String tableName,
        String partitionSpec, String uploadId) throws TunnelException;
    public void commit(Long[] blocks);
    public Long[] getBlockList();
    public String getId();
    public TableSchema getSchema();
    public UploadSession.Status getStatus();
    public Record newRecord();
    public RecordWriter openRecordWriter(long blockId);
    public RecordWriter openRecordWriter(long blockId, boolean compress);
    public RecordWriter openBufferedWriter();
    public RecordWriter openBufferedWriter(boolean compress);
}
```

Upload Objects description

- **Life cycle:** Begins with the creation of the Upload instance and ends with the completion of an upload process.
- **Create Upload instance:** An instance can be created either by Calling the Constructor or using the TableTunnel.
 - Request mode: Synchronous.
 - The server creates a session for this upload instance and a unique UploadId is generated. Obtain this ID using the getId on the client.
- **Upload data:**
 - Request mode: Synchronous.
 - Call the openRecordWriter method to generate a RecordWriter instance. The blockId identifies the data to be uploaded and indicates its location in the table within the value range [0, 20000]. If the data upload fails, use BlockId to re-upload it.
- **View upload:**
 - Request mode: Synchronous.
 - Call getStatus to obtain the current upload status.
 - Call getBlockList to obtain the successfully uploaded blockId list. Compare the result with the upload blockId list to find and re-upload failed blockIds.
- **End upload:**
 - Request mode: Synchronous.
 - Call the commit (Long[] blocks) method. The blocks list shows successfully uploaded blocks. The server verifies this list.

- This function enhances data verification. If the provided block list does not match the block list on the server, an error occurs.
- If Commit fails, try again.
- Six kinds of status are described as follows:
 - i. UNKNOWN: The initial value when the server creates a session.
 - ii. NORMAL: The upload object is created successfully.
 - iii. CLOSING: The server changes the status to CLOSING when complete is called.
 - iv. CLOSED: The upload is now complete. Precisely, moving the data to the directory where the result table is located.
 - v. EXPIRED: The upload session is timed out.
 - vi. CRITICAL: A service error has occurred.

 Note

- The blockIds in the same UploadSession must be unique. In a single UploadSession, when you use a blockId to open RecordWriter, write a batch of data, call close, and then call commit. Do not use the same blockID to open another RecordWriter to write data.
- The maximum size of a block is 100 GB, preferably more than 64 MB.
- The threshold of each session on the server is 24 hours.
- When data is being uploaded, each 8 KB of data written by the Writer triggers a network action. If no network actions are triggered within 120 seconds, the server closes the connection. In this case, open a new connection when the Writer becomes unavailable.
- We recommend that you use the openBufferedWriter interface to upload data. This interface does not show blockId details and contains an internal data cache for automatic retry upon failures. For more information, see the introductions and examples of TunnelBufferedWriter.

2.5.5. DownloadSession

This topic describes the DownloadSession interface that is used to download MaxCompute table data.

DownloadSession table data download

Definition

```
public class DownloadSession {  
    DownloadSession(Configuration conf, String projectName, String tableName,  
        String partitionSpec) throws TunnelException  
    DownloadSession(Configuration conf, String projectName, String tableName,  
        String partitionSpec, String downloadId) throws TunnelException  
    public String getId()  
    public long getRecordCount()  
    public TableSchema getSchema()  
    public TableTunnel.DownloadStatus getStatus()  
    public RecordReader openRecordReader(long start, long count)  
    public RecordReader openRecordReader(long start, long count, boolean compress)  
}
```

Description

- **Lifecycle:** This interface is used from the time when the data download instance is created to the time when the data is downloaded.
- **Create a download instance:** You can create a data download instance by calling a constructor or using TableTunnel.
 - The request mode is synchronous.
 - The server creates a session for this download instance and generates a unique ID for this instance. The instance ID can be obtained by calling getId on the client.
 - The operation has a high overhead. The server creates indexes for data files. If multiple data files exist, the operation takes a long time.
 - The server returns the total number of data records. You can start concurrent downloads based on the number of data records.
- **Download data:**
 - The request mode is asynchronous.
 - You can call openRecordReader to generate a RecordReader instance. The start parameter indicates the start position of the downloaded data record. The value of the start parameter is greater than or equal to 0. The count parameter indicates the number of downloaded data records. The value of the count parameter is greater than 0.
- **View the download status:**
 - The request mode is synchronous.
 - You can call getStatus to query the status of the data download instance.
- **The instance can be in one of the following states:**
 - **UNKNOWN:** the initial value when the server creates a session.
 - **NORMAL:** The data download instance is created.
 - **CLOSED:** The data download is complete.
 - **EXPIRED:** The download session times out.

2.5.6. TunnelBufferedWriter

To complete the uploading process, follow these steps:

1. Divide the data.
2. Specify a block ID for each data block by calling the `openRecordWriter (id)`.
3. Use one or more threads to upload the blocks. Even if a single block upload fails, you must re-upload all the blocks.
4. After uploading all blocks, provide the uploaded blockID list to the server for verification. Call `session.commit([1,2,3,...])` to complete this action.

The connection time-out and other limits on the server block manager complicate the upload process logic. So, to simplify the process, SDK provides an enhanced `RecordWriter`—`TunnelBufferWriter` interface.

This interface is defined as follows:

```
public class TunnelBufferedWriter implements RecordWriter {  
    public TunnelBufferedWriter(TableTunnel.UploadSession session, CompressOption option) throws  
    IOException;  
    public long getTotalBytes();  
    public void setBufferSize(long bufferSize);  
    public void setRetryStrategy(RetryStrategy strategy);  
    public void write(Record r) throws IOException;  
    public void close() throws IOException;  
}
```

Parameters:

- Life cycle: Begins with a `RecordWriter` creation and ends with the completion of data upload.
- Create `TunnelBufferedWriter` instance: Call `openBufferedWriter` interface of `UploadSession` to create an instance.
- Data upload: Call the `Write` interface. Data is first written to the local cache. Once the cache is full, the data is submitted to the server in batches to avoid connection time-out. Automatic retries are supported if the upload fails.
- End upload: Call the `close` interface, and then call the `Commit` interface of `UploadSession` to complete the upload process.
- Buffer control: Use the `setBufferSize` interface to modify the size of memory (bytes), occupied by the buffer preferably greater than 64 MB(default) to prevent the server from generating numerous small files that may critically impact the performance. The default value is generally used for this parameter without additional settings.
- Retry policy setting: You have three retry avoidance policies to choose from: `EXPONENTIAL_BACKOFF`, `LINEAR_BACKOFF`, and `CONSTANT_BACKOFF`. For example: The following code segment sets the number of `Write` retries to 6. To avoid unnecessary retries, each retry is performed only after exponentially ascending intervals of 4s, 8s, 16s, 32s, 64s, and 128s. This is the default configuration and generally cannot be changed.

```

RetryStrategy retry
    = new RetryStrategy(6, 4, RetryStrategy.BackoffStrategy.EXPONENTIAL_BACKOFF)
writer = (TunnelBufferedWriter) uploadSession.openBufferedWriter();
writer.setRetryStrategy(retry);

```

2.6. Bulk data channel SDK example

2.6.1. Example

This article recommends using Tunnel service address and Tunnel Buffered Writer to upload data.

- MaxCompute provides two service addresses for you to choose from. If you select the Tunnel service address, it may directly affect your data upload efficiency and billing. For more information, see [Tunnel SDK overview](#).
- We recommend that you use the TunnelBufferedWriter interface when uploading data. For more information, see the sample codes in [BufferedWriter](#).
- Operations may vary based on SDK versions. This example is provided only for your reference. Consider variances between different versions before you proceed.

2.6.2. Simple upload

```

import java.io.IOException;
import java.util.Date;
import com.aliyun.odps.Column;
import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec;
import com.aliyun.odps.TableSchema;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordWriter;
import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TunnelException;
import com.aliyun.odps.tunnel.TableTunnel.UploadSession;

public class UploadSample {
    private static String accessId = "<your access id>";
    private static String accessKey = "<your access Key>";
    private static String odpsUrl = "http://service.odps.aliyun.com/api";
    private static String tunnelUrl = "http://dt.cn-shanghai.maxcompute.aliyun-inc.com";

    //The tunnelURL must be set if you need to connect internal network, otherwise, the system uses public network as default. The example shows the Tunnel Endpoint of classical network in H
    uaDong 2, for other regions, see Access domain and data centers.

    private static String project = "<your project>";

```

```

private static String project = "<your project>";
private static String table = "<your table name>";
private static String partition = "<your partition spec>";
public static void main(String args[]) {
    Account account = new AliyunAccount(accessId, accessKey);
    Odps odps = new Odps(account);
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject(project);
    try {
        TableTunnel tunnel = new TableTunnel(odps);
        tunnel.setEndpoint(tunnelUrl); //set tunnelUrl
        PartitionSpec partitionSpec = new PartitionSpec(partition);
        UploadSession uploadSession = tunnel.createUploadSession(project,
            table, partitionSpec);
        System.out.println("Session Status is : "
            + uploadSession.getStatus().toString());
        TableSchema schema = uploadSession.getSchema();
        // After preparing data, open a Writer to start writing data. The prepared data is written
        // to one block.
        // When the data written to individual blocks is too small, the system will produce a large
        // number of small files, seriously degrading computing performance. We strongly recommend over 64
        // MB of data be written each time (up to 100 GB of data can be written to the same block).
        // You can use the average data volume and record count to estimate the total value. For
        // example: 64MB < Average data size x Record count < 100GB.
        RecordWriter recordWriter = uploadSession.openRecordWriter(0);
        Record record = uploadSession.newRecord();
        for (int i = 0; i < schema.getColumns().size(); i++) {
            Column column = schema.getColumn(i);
            switch (column.getType()) {
                case BIGINT:
                    record.setBigint(i, 1L);
                    break;
                case BOOLEAN:
                    record.setBoolean(i, true);
                    break;
                case DATETIME:
                    record.setDatetime(i, new Date());
                    break;
                case DOUBLE:
                    record.setDouble(i, 0.0);
                    break;
            }
        }
    }
}

```



```

        case STRING:
            record.setString(i, "sample");
            break;
        default:
            throw new RuntimeException("Unknown column type: "
                + column.getType());
    }
}
for (int i = 0; i < 10; i++) {
    // Writes data to the server. Each 8 KB of data written triggers a network transmiss
ion.

    // If no network transmission occurs for 120 seconds, the server closes the connect
ion. At this time, the Writer becomes unavailable and you must write data again.
    recordWriter.write(record);
}
recordWriter.close();
uploadSession.commit(new Long[] {0L});
System.out.println("upload success!");
} catch (TunnelException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

Constructor:

PartitionSpec(String spec): Uses a string to construct this class of object.

Parameters

spec: The partition definition string, such as 'pt='1',ds='2'.

In this program, the configuration must be as follows:

```
private static String partition = "pt='XXX',ds='XXX'";
```

 **Note** This paper gives the Tunnel Endpoint of East China 2 classical network. The Tunnel Endpoint settings of other regions can be referred to [Configure Endpoint](#).

2.6.3. Simple download

This topic describes how to use MaxCompute Java SDK to download data.

Download data by using the DownloadSession interface of TableTunnel

```
import java.io.IOException;
import java.util.Date;
import com.aliyun.odps.Column;
import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec;
import com.aliyun.odps.TableSchema;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordReader;
import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TableTunnel.DownloadSession;
import com.aliyun.odps.tunnel.TunnelException;
public class DownloadSample {
    private static String accessId = "<your access id>";
    private static String accessKey = "<your access Key>";
    private static String odpsUrl = "http://service.odps.aliyun.com/api";
    private static String tunnelUrl = "http://dt.cn-shanghai.maxcompute.aliyun-inc.com";
    //tunnelUrl specifies the tunnel URL. This parameter is mandatory when you download data over yo
ur intranet. If this parameter is set to null, your data is downloaded via the Internet.
    private static String project = "<your project>";
    private static String table = "<your table name>";
    private static String partition = "<your partition spec>";
    public static void main(String args[]) {
        Account account = new AliyunAccount(accessId, accessKey);
        Odps odps = new Odps(account);
        odps.setEndpoint(odpsUrl);
        odps.setDefaultProject(project);
        TableTunnel tunnel = new TableTunnel(odps);
        tunnel.setEndpoint(tunnelUrl);//Set tunnelUrl.
        PartitionSpec partitionSpec = new PartitionSpec(partition);
        try {
            DownloadSession downloadSession = tunnel.createDownloadSession(project, table,
                partitionSpec);
            System.out.println("Session Status is : "
                + downloadSession.getStatus().toString());
            long count = downloadSession.getRecordCount();
            System.out.println("RecordCount is: " + count);
            RecordReader recordReader = downloadSession.openRecordReader(0,
```

```

        count);
    Record record;
    while ((record = recordReader.read()) != null) {
        consumeRecord(record, downloadSession.getSchema());
    }
    recordReader.close();
} catch (TunnelException e) {
    e.printStackTrace();
} catch (IOException e1) {
    e1.printStackTrace();
}
}

private static void consumeRecord(Record record, TableSchema schema) {
    for (int i = 0; i < schema.getColumns().size(); i++) {
        Column column = schema.getColumn(i);
        String colValue = null;
        switch (column.getType()) {
            case BIGINT: {
                Long v = record.getBigint(i);
                colValue = v == null ? null : v.toString();
                break;
            }
            case BOOLEAN: {
                Boolean v = record.getBoolean(i);
                colValue = v == null ? null : v.toString();
                break;
            }
            case DATETIME: {
                Date v = record.getDatetime(i);
                colValue = v == null ? null : v.toString();
                break;
            }
            case DOUBLE: {
                Double v = record.getDouble(i);
                colValue = v == null ? null : v.toString();
                break;
            }
            case STRING: {
                String v = record.getString(i);
                colValue = v == null ? null : v.toString();
                break;
            }
        }
    }
}

```

```
        break;
    }
    default:
        throw new RuntimeException("Unknown column type: "
            + column.getType());
    }
    System.out.print(colValue == null ? "null" : colValue);
    if (i != schema.getColumns().size())
        System.out.print("\t");
    }
    System.out.println();
}
}
```

 Note

- In the preceding command, a tunnel endpoint on the classic network in the China East 2 (Shanghai) region is used as an example. For more information about how to configure tunnel endpoints in the other regions, see [Access domains and data centers](#).
- To make the test easier, we use `System.out.printl` to output data. You can choose to output data as a file in TXT format.

Download data by using the DownloadSession interface of InstanceTunnel

```
Odps odps = OdpsUtils.newDefaultOdps(); //Initialize Open Data Processing Service (ODPS) objects.
Instance i = SQLTask.run(odps, "select * from wc_in;");
i.waitForSuccess();

//Create an instance tunnel.
InstanceTunnel tunnel = new InstanceTunnel(odps);
//Create a download session based on the specified instance ID.
InstanceTunnel.DownloadSession session = tunnel.createDownloadSession(odps.getDefaultProject(), i.getId());

long count = session.getRecordCount();
//Specify the number of records that will be presented.
System.out.println(count);

//Obtain data by using the same method as you do with TableTunnel.
TunnelRecordReader reader = session.openRecordReader(0, count);
Record record;
while ((record = reader.read()) != null) {
    for (int col = 0; col < session.getSchema().getColumns().size(); ++col) {
        //Specify that all the fields in the wc_in table are strings and will be directly printed.
        System.out.println(record.get(col));
    }
}
reader.close();
```

Download data by using SQLTask.getResultSet()

```
Odps odps = OdpsUtils.newDefaultOdps(); //Initialize ODPS objects.
Instance i = SQLTask.run(odps, "select * from wc_in;");
i.waitForSuccess();

//Obtain the result iterator based on the specified instance object.
ResultSet rs = SQLTask.getResultSet(i);
for (Record r : rs) {
    //Specify the number of records that will be presented.
    System.out.println(rs.getRecordCount());

    for (int col = 0; col < rs.getTableSchema().getColumns().size(); ++col) {
        //Specify that all the fields in the wc_in table are strings and will be directly printed.
        System.out.println(r.get(col));
    }
}
```

2.6.4. Example for multi-thread uploading

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import com.aliyun.odps.Column;
import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec;
import com.aliyun.odps.TableSchema;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordWriter;
import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TunnelException;
import com.aliyun.odps.tunnel.TableTunnel.UploadSession;
class UploadThread implements Callable<Boolean> {
    private long id;
    private RecordWriter recordWriter;
    private Record record;
    private TableSchema tableSchema;
```

```
public UploadThread(long id, RecordWriter recordWriter, Record record,
    TableSchema tableSchema) {
    this.id = id;
    this.recordWriter = recordWriter;
    this.record = record;
    this.tableSchema = tableSchema;
}
@Override
public Boolean call() {
    for (int i = 0; i < tableSchema.getColumns().size(); i++) {
        Column column = tableSchema.getColumn(i);
        switch (column.getType()) {
            Case bigint:
                record.setBigint(i, 1L);
                Break;
            Case Boolean:
                record.setBoolean(i, true);
                break;
            case DATETIME:
                record.setDatetime(i, new Date());
                break;
            case DOUBLE:
                record.setDouble(i, 0.0);
                break;
            case STRING:
                record.setString(i, "sample");
                break;
            default:
                throw new RuntimeException("Unknown column type: "
                    + column.getType());
        }
    }
    for (int i = 0; i < 10; i++) {
        try {
            recordWriter.write(record);
        } catch (IOException e) {
            recordWriter.close();
            e.printStackTrace();
            return false;
        }
    }
}
```

```

        recordWriter.close();
        return true;
    }
}

public class UploadThreadSample {
    private static String accessId = "<your access id>";
    private static String accessKey = "<your access Key>";
    private static String odpsUrl = "<http://service.odps.aliyun.com/api>";
    private static String tunnelUrl = "<http://dt.cn-shanghai.maxcompute.aliyun-inc.com>";

    //The tunnelURL must be set if you need to connect internal network, otherwise, the system uses public network as default. The example shows the Tunnel Endpoint of classical network in H
    uaDong 2, for other regions, see Access domain and data centers.

    private static String project = "<your project>";
    private static String table = "<your table name>";
    private static String partition = "<your partition spec>";
    private static int threadNum = 10;
    public static void main(String args[]) {
        Account account = new AliyunAccount(accessId, accessKey);
        Odps odps = new Odps(account);
        odps.setEndpoint(odpsUrl);
        odps.setDefaultProject(project);
        try {
            TableTunnel tunnel = new TableTunnel(odps);
            tunnel.setEndpoint(tunnelUrl); //set tunnelUrl
            PartitionSpec partitionSpec = new PartitionSpec(partition);
            UploadSession uploadSession = tunnel.createUploadSession(project,
                table, partitionSpec);
            System.out.println("Session Status is : "
                + uploadSession.getStatus().toString());
            ExecutorService pool = Executors.newFixedThreadPool(threadNum);
            ArrayList<Callable<Boolean>> callers = new ArrayList<Callable<Boolean>>();
            for (int i = 0; i < threadNum; i++) {
                RecordWriter recordWriter = uploadSession.openRecordWriter(i);
                Record record = uploadSession.newRecord();
                callers.add(new UploadThread(i, recordWriter, record,
                    uploadSession.getSchema()));
            }
            pool.invokeAll(callers);
            pool.shutdown();
            Long[] blockList = new Long[threadNum];

```



```

        for (int i = 0; i < threadNum; i++)
            blockList[i] = Long.valueOf(i);
        uploadSession.commit(blockList);
        System.out.println("upload success!") ;
    } catch (TunnelException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}

```

The Tunnel Endpoint can be specified or left blank.

- If specified, the uploading data goes through the specified Endpoint.
- If not specified, the uploading data goes through public network.
- This paper gives the Tunnel Endpoint of East China 2 classical network. The Tunnel Endpoint settings of other regions can be referred to [Configure Endpoint](#).

2.6.5. Example for multi-thread downloading

This article shows you how to use the TableTunnel interface to achieve multithreaded download through code examples.

```

import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import com.aliyun.odps.Column;
import com.aliyun.odps.Odps;
import com.aliyun.odps.PartitionSpec;
import com.aliyun.odps.TableSchema;
import com.aliyun.odps.account.Account;
import com.aliyun.odps.account.AliyunAccount;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.RecordReader;
import com.aliyun.odps.tunnel.TableTunnel;

```

```
import com.aliyun.odps.tunnel.TableTunnel;
import com.aliyun.odps.tunnel.TableTunnel.DownloadSession;
import com.aliyun.odps.tunnel.TunnelException;
class DownloadThread implements Callable<Long> {
    private long id;
    private RecordReader recordReader;
    private TableSchema tableSchema;
    public DownloadThread(int id,
        RecordReader recordReader, TableSchema tableSchema) {
        this.id = id;
        this.recordReader = recordReader;
        this.tableSchema = tableSchema;
    }
    @Override
    public Long call() {
        Long recordNum = 0L;
        try {
            Record record;
            while ((record = recordReader.read()) != null) {
                recordNum++;
                System.out.print("Thread " + id + "\t");
                consumeRecord(record, tableSchema);
            }
            recordReader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return recordNum;
    }
    private static void consumeRecord(Record record, TableSchema schema) {
        for (int i = 0; i < schema.getColumns().size(); i++) {
            Column column = schema.getColumn(i);
            String colValue = null;
            switch (column.getType()) {
                case BIGINT: {
                    Long v = record.getBigint(i);
                    colValue = v == null ? null : v.toString();
                    Break;
                }
                case BOOLEAN: {
                    Boolean v = record.getBoolean(i);
```

```

        colValue = v == null ? null : v.toString();
        break;
    }
    case DATETIME: {
        Date v = record.getDatetime(i);
        colValue = v == null ? null : v.toString();
        break;
    }
    case DOUBLE: {
        Double v = record.getDouble(i);
        colValue = v == null ? null : v.toString();
        break;
    }
    case STRING: {
        String v = record.getString(i);
        colValue = v == null ? null : v.toString();
        break;
    }
    Default:
        throw new RuntimeException("Unknown column type: "
            + column.getType());
    }
    System.out.print(colValue == null ? "null" : colValue);
    If (!= schema.getColumns().size())
        System.out.print("\t");
}
System.out.println();
}
}

public class DownloadThreadSample {
    private static String accessId = "<your access id>";
    private static String accessKey = "<your access Key>";
    private static String odpsUrl = "http://service.odps.aliyun.com/api";
    private static String tunnelUrl = "http://dt.cn-shanghai.maxcompute.aliyun-inc.com";

    //The tunnelURL must be set if you need to connect internal network, otherwise, the system uses public network as default. The example shows the Tunnel Endpoint of classical network in H
    uaDong 2, for other regions, see Access domain and data centers.

    private static String project = "<your project>";
    private static String table = "<your table name>";
    private static String partition = "<your partition spec>";
    private static int threadNum = 10;

```

```

public static void main(String args[]) {
    Account account = new AliyunAccount(accessId, accessKey);
    Odps odps = new Odps(account);
    odps.setEndpoint(odpsUrl);
    odps.setDefaultProject(project);
    TableTunnel tunnel = new TableTunnel(odps);
    tunnel.setEndpoint(tunnelUrl); //set tunnelUrl
    PartitionSpec partitionSpec = new PartitionSpec(partition);
    DownloadSession downloadSession;
    try {
        downloadSession = tunnel.createDownloadSession(project, table,
            partitionSpec);
        System.out.println("Session Status is : "
            + downloadSession.getStatus().toString());
        long count = downloadSession.getRecordCount();
        System.out.println("RecordCount is: " + count);
        ExecutorService pool = Executors.newFixedThreadPool(threadNum);
        ArrayList<Callable<Long>> callers = new ArrayList<Callable<Long>>();
        long start = 0;
        long step = count / threadNum;
        for (int i = 0; i < threadNum - 1; i++) {
            RecordReader recordReader = downloadSession.openRecordReader(
                step * i, step);
            callers.add(new DownloadThread( i, recordReader, downloadSession.getSchema()))
;
        }
        RecordReader recordReader = downloadSession.openRecordReader(step * (threadNum -
1), count
        - ((threadNum - 1) * step));
        callers.add(new DownloadThread( threadNum - 1, recordReader, downloadSession.getS
chema()));
        Long downloadNum = 0L;
        List<Future<Long>> recordNum = pool.invokeAll(callers);
        for (Future<Long> num : recordNum)
            downloadNum += num.get();
        System.out.println("Record Count is: " + downloadNum);
        pool.shutdown();
    } catch (TunnelException e) {
        e.printStackTrace();
    } catch (IOException e) {

```

```
        e.printStackTrace();
    } catch (InterruptedException e) {
        e.printStackTrace();
    } catch (ExecutionException e) {
        e.printStackTrace();
    }
}
}
```

Note

The Tunnel Endpoint can be specified or left blank.

- If specified, the downloading data goes through the specified Endpoint.
- If not specified, the downloading data goes through public network Endpoint.
- This paper gives the Tunnel Endpoint of East China 2 classical network. The Tunnel Endpoint settings of other regions can be referred to [Configure Endpoint](#).

2.6.6. Example for BufferedWriter uploading

This article shows you how to use the BufferedWriter interface to upload data through code examples.

```
// Initializes MaxCompute and Tunnel code
RecordWriter writer = null;
TableTunnel.UploadSession uploadSession = tunnel.createUploadSession(projectName, tableName);
try {
    int i = 0;
    // Generates TunnelBufferedWriter instance
    writer = uploadSession.openBufferedWriter();
    Record product = uploadSession.newRecord();
    for (String item : items) {
        product.setString("name", item);
        product.setBigint("id", i);
        // Calls the Write interface to write data
        writer.write(product);
        i += 1;
    } finally {
        if (writer != null) {
            // Closes TunnelBufferedWriter
            writer.close();
        }
    }

    // Submits data via uploadSession to end the upload process
    uploadSession.commit();
}
```

2.6.7. Example for BufferedWriter multi-thread uploading

This article shows you how to use BufferedWriter interface to realize multithreaded upload through code examples.

```
class UploadThread extends Thread {
    private UploadSession session;
    private static int RECORD_COUNT = 1200;
    public UploadThread(UploadSession session) {
        this.session = session;
    }
    @Override
    public void run () {
        RecordWriter writer = up.openBufferedWriter();
        Record r = up.newRecord();
        for (int i = 0; i < RECORD_COUNT; i++) {
            r.setBigint(0, i);
            writer.write(r);
        }
        writer.close();
    }
};

public class Example {
    public static void main(String args[]) {
        // Initializes MaxCompute and Tunnel code
        TableTunnel.UploadSession uploadSession = tunnel.createUploadSession(projectName, tableName);
        UploadThread t1 = new UploadThread(up);
        UploadThread t2 = new UploadThread(up);
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        uploadSession.commit();
    }
}
```

2.7. Import or export data by using Data Integration

This topic describes how to import and export MaxCompute data by using DataWorks Data Integration.

Data Integration import and export

Context

Data Integration in DataWorks supports the MaxCompute data source. It allows you to write data from other data sources to MaxCompute or write MaxCompute data to other data sources. Leveraging the underlying Tunnel feature, Data Integration achieves the MaxCompute data read and write functions.

You can import and export data in wizard or script mode. This topic details data import and export in wizard mode.

 **Note** For more information about data import and export in script mode, see [Create a sync node by using the code editor](#).

Procedure

1. On the **Data Integration** page, configure the MaxCompute data source. For more information, see [Configure a MaxCompute connection](#).
2. On the **Data Analytics** page, create a business flow. For more information, see [Create a workflow](#).
3. [Create an offline data synchronization node](#).
4. [Select a data source](#).
 - To export data from MaxCompute, select MaxCompute as the data source. For more information, see [Configure MaxCompute Reader](#).
 - To import data to MaxCompute, select another data source. For more information, see [Configure Reader plug-ins](#).
5. [Select a data destination](#).
 - To import data to MaxCompute, select MaxCompute as the data destination.
 - To export data from MaxCompute, select another data source as the data destination.
6. [Map the fields in the source and destination tables](#).
7. [Configure channel control policies](#).
8. [Configure the node properties](#).
9. Save the configurations and click .

3.SQL

3.1. SQL overview

This topic describes MaxCompute SQL keywords, data type conversions, partitioned tables, UNION ALL operations, and limits.

keyword data type conversion partitioned table UNION ALL

Introduction

MaxCompute SQL is used for offline batch computing and computing scenarios that involve gigabytes, terabytes, or exabytes of data. After you submit a MaxCompute job, it is scheduled in dozens of seconds or several minutes. Therefore, MaxCompute is suitable for batch jobs that process large volumes of data. It is not suitable for frontend business systems that process several thousand or tens of thousands of transactions per second.

The MaxCompute SQL syntax is similar to the SQL syntax. The MaxCompute SQL syntax is a subset of the standard ANSI SQL-92 syntax but it also extends the standard syntax. MaxCompute projects lack many database features, such as transactions, primary key constraints, and indexes. Therefore, MaxCompute projects cannot be regarded as databases. The maximum size of an SQL statement supported by MaxCompute is 2 MB.

Keywords

The keywords of SQL statements are reserved words in MaxCompute. If you use keywords to name tables, columns, or partitions, you need to escape the keywords by using a pair of backticks (``). Otherwise, an error is returned. Reserved words are not case-sensitive. The following section lists common reserved words. For a complete list of reserved words, see [Reserved words and keywords](#).

```
% & && ( ) * +
- . / ; < <= <>
= > >= ? ADD ALL ALTER
AND AS ASC BETWEEN BIGINT BOOLEAN BY
CASE CAST COLUMN COMMENT CREATE DESC DISTINCT
DISTRIBUTE DOUBLE DROP ELSE FALSE FROM FULL
GROUP IF IN INSERT INTO IS JOIN
LEFT LIFECYCLE LIKE LIMIT MAPJOIN NOT NULL
ON OR ORDER OUTER OVERWRITE PARTITION RENAME
REPLACE RIGHT RLIKE SELECT SORT STRING TABLE
THEN TOUCH TRUE UNION VIEW WHEN WHERE
```

Data type conversions

MaxCompute SQL supports two modes of data type conversions: explicit conversion and implicit conversion. For more information, see [类型转换](#).

- An explicit conversion uses CAST to convert the data type of a value.

- An implicit conversion is automatically performed by MaxCompute based on the context and a set of predefined rules. The implicit conversion scope includes various operators and built-in functions.

Partitioned tables

MaxCompute SQL supports partitioned tables, which offers you higher SQL query performance and reduces your costs. For more information about partitions, see [Partition](#).

UNION ALL

Data types of content in columns, the number of columns, and column names used in a UNION ALL operation must be the same. Otherwise, an error is returned.

Limits

For limits of MaxCompute SQL, see [MaxCompute SQL limits](#). For more information about DDL and DML statements that are not supported by MaxCompute, see [Differences with other SQL statements](#).

Other limits:

- For more information about the limits of scalar subqueries, see [Scalar subquery](#).
- If you execute an INSERT statement to insert values, the values must be constants.
- MaxCompute allows you to perform a UNION ALL or UNION operation on a maximum of 256 tables.
- The size of small tables on which you perform a MAPJOIN operation cannot exceed 512 MB.
- Due to the adjustment made by the International Organization for Standardization (ISO) on the UTC+8 time zone, the DATETIME data obtained by executing related SQL statements is different from the actual time in certain periods. For the DATETIME data between the year 1900 and the year 1928, the time difference is 352 seconds. For the DATETIME data before the year 1900, the time difference is 9 seconds.

3.2. MaxCompute SQL limits

This topic describes all the limits of MaxCompute SQL statements.

Item	Maximum value/Limit	Category	Description
Table name length	128 bytes	Length limit	A table or column name cannot contain special characters. It must start with a letter, and can contain only letters, digits, and underscores (_).
Comment length	1,024 bytes	Length limit	A comment is a valid string that can be up to 1,024 bytes in length.
Columns in a table	1,200	Quantity limit	A table can contain a maximum of 1,200 columns.
Partitions in a table	60,000	Quantity limit	A table can contain a maximum of 60,000 partitions.

Item	Maximum value/Limit	Category	Description
Partition levels of a table	6	Quantity limit	A table can contain a maximum of six levels of partitions.
Screen display	10,000 rows	Quantity limit	A SELECT statement can return a maximum of 10,000 rows.
INSERT targets	256	Quantity limit	A MULTI-INSERT operation can insert data to up to 256 tables at a time.
UNION ALL targets	256	Quantity limit	A UNION ALL operation can merge up to 256 tables at a time.
MAPJOIN targets	128	Quantity limit	A MAPJOIN operation can join up to 128 small tables at a time.
MAPJOIN memory	512 MB	Size limit	The memory size for all small tables on which the MAPJOIN operation is performed cannot exceed 512 MB.
Window functions	5	Quantity limit	A SELECT statement can contain a maximum of five window functions.
ptinsubq	1,000 rows	Quantity limit	If a subquery contains a partition column, the subquery can return no more than 1,000 rows.
Length of an SQL statement	2 MB	Length limit	The maximum length of an SQL statement is 2 MB. This limit is suitable for the scenarios where you use the SDK to call SQL statements.
Conditions of a WHERE clause	256	Quantity limit	A WHERE clause can contain a maximum of 256 conditions.
Length of a column record	8 MB	Length limit	The maximum length of a column record in a table is 8 MB.

Item	Maximum value/Limit	Category	Description
Parameters in an IN clause	1024	Quantity limit	This item specifies the maximum number of parameters in an IN clause, for example, <code>in (1,2,3...,1024)</code> . If an <code>in (...)</code> clause contains excessive parameters, the compilation performance will be affected. We recommend that you use no more than 1,024 parameters, but this is not a fixed upper limit.
<code>jobconf.json</code>	1 MB	Size limit	The maximum size of the <code>jobconf.json</code> file is 1 MB. If a table contains a large number of partitions, the size of the <code>jobconf.json</code> file may exceed 1 MB.
View	Not writable	Operation limit	A view is not writable and does not support the <code>INSERT</code> operation.
Data type and position of a column	Unmodifiable	Operation limit	The data type and position of a column are unmodifiable.
Java UDFs	Not allowed to be <code>abstract</code> or <code>static</code>	Operation limit	Java UDFs cannot be <code>abstract</code> or <code>static</code> .
Partitions that can be queried	10,000	Quantity limit	A maximum of 10,000 partitions can be queried.
SQL execution plans	1 MB	Size limit	The size of an execution plan generated from MaxCompute SQL statements cannot exceed 1 MB. Otherwise, the error message <code>FAILED: ODPS-0010000:System internal error - The Size of Plan is too large</code> appears.

 **Note** The preceding MaxCompute SQL limits cannot be manually modified.

3.3. Differences with other SQL statements

This topic lists SQL statements that are supported by MaxCompute by comparing MaxCompute SQL statements with SQL statements of Hive, MySQL, Oracle, and SQL Server.

DDL statement DML statement SCRIPTING statement

Support for DDL statements

Statement	MaxCompute	Hive	MySQL	Oracle	SQL Server
CREATE TABLE—PRIMARY KEY	N	N	Y	Y	Y
CREATE TABLE—NOT NULL	N	N	Y	Y	Y
CREATE TABLE—CLUSTER BY	Y	Y	N	Y	Y
CREATE TABLE—EXTERNAL TABLE	Y (OSS, OTS, TDDL)	Y	N	Y	N
CREATE TABLE—TEMPORARY TABLE	N	Y	Y	Y	Y (with the # prefix)
INDEX—CREATE INDEX	N	Y	Y	Y	Y
VIRTUAL COLUMN	N	N	N	Y	Y

Support for DML statements

Statement	MaxCompute	Hive	MySQL	Oracle	SQL Server
CTE	Y	Y	Y	Y	Y
SELECT—recursive CTE	N	N	N	Y	Y
SELECT—GROUP BY ROLL UP	Y	Y	Y	Y	Y
SELECT—GROUP BY CUBE	Y	Y	N	Y	Y
SELECT—GROUPING SET	Y	Y	N	Y	Y

Statement	MaxCompute	Hive	MySQL	Oracle	SQL Server
SELECT— IMPLICIT JOIN	Y	Y	N	Y	Y
SELECT— PIVOT	N	N	N	Y	Y
SEMI JOIN	Y	Y	Y	N	N
SELECT TRANSFO RM	Y	Y	N	N	N
SELECT— correlate d subquery	Y	Y	Y	Y	Y
ORDER BY NULLS FIRST/LA ST	Y	Y	Y	Y	Y
LATERAL VIEW	Y	Y	N	Y	Y (CROSS APPLY keyword)
SET OPERATO R—UNION (distinct)	Y	Y	Y	Y	Y
SET OPERATO R— INTERSEC T	Y	N	N	Y	Y
SET OPERATO R— MINUS/E XCEPT	Y	N	N	Y	Y (keyword EXCEPT)
INSERT INTO ... VALUES	Y	Y	Y	Y	Y
INSERT INTO (ColumnL ist)	Y	Y	Y	Y	Y

Statement	MaxCompute	Hive	MySQL	Oracle	SQL Server
UPDATE ... WHERE	N	Y	Y	Y	Y
UPDATE ... ORDER BY LIMIT	N	N	Y	N	Y
DELETE ... WHERE	N	Y	Y	Y	Y
DELETE ... ORDER BY LIMIT	N	N	Y	N	N
ANALYTIC— reusable WINDOWING CLAUSE	Y	Y	N	N	N
ANALYTIC— CURRENT ROW	Y	Y	N	Y	Y
ANALYTIC— UNBOUNDED	Y	N	Y	Y	Y
ANALYTIC— RANGE ...	N	Y	N	Y	Y
WHILE DO	N	N	Y	Y	Y

Support for SCRIPTING statements

Statement	MaxCompute	Hive	MySQL	Oracle	SQL Server
TABLE VARIABLE	Y	Y	Y	Y	Y
SCALAR VARIABLE	Y	Y	Y	Y	Y
ERROR HANDLING— RAISE ERROR	N	N	Y	Y	Y

Statement	MaxCompute	Hive	MySQL	Oracle	SQL Server
ERROR HANDLING—TRY CATCH	N	N	N	Y	Y
FLOW CONTROL—LOOP	N	N	Y	Y	Y
CURSOR	N	N	Y	Y	Y

3.4. Operators

Operators are used to perform program code operations. This article introduces four types of operators: relational operator, arithmetic operator, bit operator and logical operator.

Relational operators

Operator	Description
A=B	If A or B is NULL, NULL is returned. If A is equal to B, TRUE is returned; otherwise FALSE is returned.
A<>B	If A or B is NULL, NULL is returned. If A is not equal to B, TRUE is returned; otherwise FALSE is returned.
A<B	If A or B is NULL, NULL is returned. If A is less than B, TRUE is returned; otherwise FALSE is returned.
A<=B	If A or B is NULL, NULL is returned. If A is not greater than B, TRUE is returned; otherwise FALSE is returned.
A>B	If A or B is NULL, NULL is returned. If A is greater than B, TRUE is returned; otherwise FALSE is returned.
A>=B	If A or B is NULL, NULL is returned; if A is not less than B, TRUE is returned; otherwise, FALSE is returned.
A IS NULL	If A is NULL, TRUE is returned; otherwise, FALSE is returned.
A IS NOT NULL	If A is NULL, TRUE is returned; otherwise FALSE is returned.

Operator	Description
A LIKE B	If A or B is NULL, NULL is returned. If String A matches the SQL simple regular B TRUE is returned; otherwise FALSE is returned. The (%) character in B matches an arbitrary number of characters and the (_) character in B matches any character in A. To match (%) or(_), use by the escape characters ('%') and ('_'). <pre> 'aaa' like 'a_' = TRUE 'aaa' like 'a%' = TRUE 'aaa' like 'aab' = FALSE 'a%b' like 'a\\%b' = TRUE 'axb' like 'a\\%b' = FALSE </pre>
A RLIKE B	A is a string, and B is a string constant regular expression. If any substring of A matches the java regular expression B, TRUE is returned; otherwise FALSE is returned. If expression B is empty, report an error and exit. If expression A or B is NULL, NULL is returned.
A IN B	B is a set. If expression A is NULL, NULL is returned. If expression A is in expression B, TRUE is returned; otherwise FALSE is returned. If expression B has only one element NULL, that is, A IN (NULL), return NULL. If expression B contains NULL element, take NULL as the type of other elements in B set. B must be a constant and at least has one element; all types must be consistent.
BETWEEN AND	The expression is <code>A [NOT] BETWEEN B AND C</code> . Empty if A, B, or C is empty. TRUE if A is larger than or equal to B and less than or equal to C; otherwise FALSE is returned.

The common use:

```

select * from user where user_id = '0001';
select * from user where user_name <> 'maggie';
select * from user where age > '50';
select * from user where birth_day >= '1980-01-01 00:00:00';
select * from user where is_female is null;
select * from user where is_female is not null;
select * from user where user_id in (0001,0010);
select * from user where user_name like 'M%';

```

The Double values in MaxCompute are different in precision. For this reason, we do not recommend using the equal sign for comparison between two Double data. You can subtract two Double types, and then take the absolute value into consideration. When the absolute value is small enough, the two double values are considered equal.

Example:

```
abs(0.9999999999 - 1.0000000000) < 0.0000000001
```

-- 0.9999999999 and 1.0000000000 have the precision of 10 decimal digits, while 0.0000000001 has the precision of 9 decimal digits.

-- It is considered that 0.9999999999 is equal to 1.0000000000.

Note

- ABS is a built-in function provided by MaxCompute to take absolute value. For more information, see [ABS](#).
- In general, the DOUBLE type in MaxCompute can retain 14-bit decimal.

Arithmetic operators

Operator	Description
A + B	If expression A or B is NULL, NULL is returned; otherwise the result of A+B is returned.
A - B	If expression A or B is NULL, NULL is returned; otherwise the result of A - B is returned.
A * B	If expression A or B is NULL, NULL is returned; otherwise result of A * B is returned.
A / B	If expression A or B is NULL, NULL is returned; otherwise the result of A / B is returned. If Expression A and B are BIGINT types, the result is DOUBLE type.
A % B	If expression A or B is NULL, NULL is returned; otherwise the reminder result from dividing A by B is returned.
+A	Result A is returned.
-A	If expression A is NULL, NULL is returned; otherwise -A is returned.

The common use:

```
select age+10, age-10, age%10, -age, age*age, age/10 from user;
```

 Note

- You can only use **STRING**, **BIGINT**, and **DOUBLE** to perform arithmetic operations. (Using **Datetime** type and **Boolean** type is restricted.)
- Before you begin these operations, the type **STRING** is converted into **DOUBLE** by implicit type conversion.
- If **BIGINT** and **DOUBLE** both are involved in arithmetic operation, the type **BIGINT** is converted into **DOUBLE** by implicit type conversion.
- When **A** and **B** are **BIGINT** types, the return result of **A/B** will be a **DOUBLE** type. For other arithmetic operations, the return value is also a **BIGINT** type.

Bitwise operators

Operator	Description
A & B	Return the result of bitwise AND of A and B. For example: 1&2, return 0; 1&3, return 1; Bitwise AND of NULL and other values, all return NULL. Expression A and B must be BIGINT .
A B	Return the result of bitwise OR of A and B. For example: 1 2, return 3. 1 3, return 3. Bitwise OR of NULL and other values, all return NULL. Expression A and B must be BIGINT type.

 **Note** Bitwise operator does not support implicit conversions, only supports the type **BIGINT**.

Logical operators

Operator	Description
A and B	TRUE and TRUE=TRUE
	TRUE and FALSE=FALSE
	FALSE and TRUE=FALSE
	FALSE and NULL=FALSE
	NULL and FALSE=FALSE
	TRUE and NULL=NULL
	NULL and TRUE=NULL
	NULL and NULL=NULL
	TRUE or TRUE=TRUE
	TRUE or FALSE=TRUE

Operator	Description
A or B	FALSE or TRUE=TRUE
	FALSE or NULL=NULL
	NULL or FALSE=NULL
	TRUE or NULL=TRUE
	NULL or TRUE=TRUE
	NULL or NULL=NULL
NOT A	If A is NULL, NULL is returned.
	If A is TRUE, FALSE is returned.
	If A is FALSE, TRUE is returned.

 **Note** Only the type Boolean can be involved in logic operations and the implicit type conversion is not supported.

3.5. DDL SQL

3.5.1. Table-level operations

This topic describes how to run statements to create, view, delete, rename, and modify tables on the MaxCompute client.

Create a table

Syntax:

```

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
[(col_name data_type [DEFAULT value] [COMMENT col_comment], ...)]
[COMMENT table_comment]
[PARTITIONED BY (col_name data_type [COMMENT col_comment], ...)]
[CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [ASC | DESC] [, col_name [ASC | DESC] .
..))] INTO number_of_buckets BUCKETS] -- Used to set the shuffle and sort attributes when you create
a hash cluster table.
[STORED BY StorageHandler] -- Used only for external tables.
[WITH SERDEPROPERTIES (Options)] -- Used only for external tables.
[LOCATION OSSLocation]; -- Used only for external tables.
[LIFECYCLE days]
[AS select_statement]
CREATE TABLE [IF NOT EXISTS] table_name
LIKE existing_table_name

```

- Table names and column names cannot contain special characters. The names can contain lowercase letters (a-z), uppercase letters (A-Z), digits, and underscores (_), and must start with a letter.
- When a table is created, an error is returned if another table has the same name, but the `IF NOT EXISTS` option is not specified. If the option is specified, no matter whether another table that has the same name exists or even if the source table structure and the target table structure are inconsistent, a success message is returned. The meta information of the existing table that has the same name remains unchanged.
- Both the table name and column name are not case-sensitive and cannot contain special characters. A table name or column name can contain only lowercase and uppercase letters, digits, and underscores (_). We recommend that you start the name with a letter. Neither of them can exceed 128 bytes in length. You can define a maximum of 1,200 columns in a single table.
- MaxCompute 2.0 supports many **data types**, including BIGINT, DOUBLE, BOOLEAN, DATETIME, DECIMAL, and STRING.

Note If new data types such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, and BINARY are used in an SQL statement, use the `set` statement to enable the new data type flag:

- Session level: To use a new data type, add a `set` statement `set odps.sql.type.system.odps2=true;` before the SQL statement and submit the two statements simultaneously.
- Project level: New data types at the project level can be used. To use the new data types at the project level, the project owner must run the following command:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Other operations](#). For precautions on enabling new data types for a project, see [Data types](#).

- Use `DEFAULT value` to specify the default value of a column. If the INSERT statement does not specify the value of the column, the default value is used.
- Use `PARTITIONED BY` to specify the column on which a table is partitioned. For more information, see [Partition](#). MaxCompute 2.0 supports the following data types for partition columns: TINYINT, SMALLINT, INT, BIGINT, VARCHAR, and STRING.

The value of the partition field cannot have a double-byte character and must start with a letter, followed by a letter or number. The name cannot exceed 128 bytes in length. Special characters can be used, including spaces, colons (:), underscores (_), dollar signs (\$), pound signs (#), periods (.), exclamation points (!), and at signs (@). Other characters, such as the tab (\t), new line (\n), and forward slashes (/), are considered as undefined characters. If the partition field is used to partition a table, a full scan is not needed when you add a partition or update or read data in a partition. This makes table processing more efficient.

Note A maximum of 60,000 partitions for six or lower levels are allowed in a table.

- The content of a comment is a string whose length does not exceed 1024 bytes.
- `LIFECYCLE` indicates the lifecycle (in days) of a table. The `CREATE TABLE LIKE` statement does not duplicate the lifecycle attribute from the source table.
- Use `CLUSTERED BY` to specify hash keys. MaxCompute performs the hash operation on the specified columns and distributes data to each bucket based on the hash values.
 - To prevent data skew and hot spots and to better run concurrent statements, we recommend that you specify columns with a large value range and a small number of duplicate key values in `CLUSTERED BY`. In addition, to optimize the `JOIN` operation, we recommend that you select commonly used join or aggregation keys. Join and aggregation keys are similar to primary keys in conventional databases.
 - `SORTED BY` specifies how to sort fields in a bucket. We recommend that you keep the value of `SORTED BY` consistent with that of `CLUSTERED BY` to make performance higher. After you specify fields in the `SORTED BY` clause, MaxCompute automatically generates indexes. It also executes the clause faster based on the indexes when you query data.

- `INTO number_of_buckets BUCKETS` specifies the number of hash buckets. The value of this field varies with the data volume. By default, MaxCompute supports a maximum of 1,111 reducers. Therefore, a maximum of 1,111 hash buckets are allowed. You can run the `set odps.sql.reducer.instances=xxx;` command to increase the maximum value (4,000) of this field. Performance deteriorates if the maximum value exceeds 4,000.

We recommend that you abide by the following principles when you specify the number of hash buckets:

- The recommended size of each bucket is around 500 MB. If you want to add 1,000 buckets to a 500-GB partition, the size of each bucket is 500 MB on average. If a table contains large amounts of data, you can increase the size of each bucket from 500 MB to 2 or 3 GB. You can also run the `set odps.sql.reducer.instances=xxx` command to increase the number of hash buckets beyond 1,111.
- If two tables are frequently joined, specify the same number of hash buckets for them to better execute `join` statements and to skip Shuffle and Sort. If the number of hash buckets for one table is different from that for the other table, we recommend that you use the greater number for both tables. This guarantees that SQL statements can be run concurrently in an efficient manner.
- Hash cluster tables help optimize the following aspects:
 - Bucket pruning
 - Aggregation
 - Storage
- Hash cluster tables are subject to the following limits:
 - The `INSERT INTO` statement is not supported. You can add data only by using the `INSERT OVERWRITE` statement.
 - Data cannot be uploaded to a range cluster table by using Tunnel because data uploaded by using Tunnel is unordered.
- For more information about external tables, see [Access OSS data by using the built-in extractor](#).

If you want to create a partitioned table named `sale_detail` to store sales records and use `sale_date` and `region` as partition key columns, you can use the following statement:

```
create table if not exists sale_detail
(
  shop_name  string,
  customer_id string,
  total_price double
)
partitioned by (sale_date string,region string);
-- Create a partition table named sale_detail.
```

The statement `create table...as select...` can also be used to create a table. After a table is created, the data is copied to the new table. Example:

```
create table sale_detail_ctas1 as
select * from sale_detail;
```

If the `sale_detail` table has data, the abovementioned example copies all data of `sale_detail` into the `sale_detail_ctas1` table.

 **Note** The `sale_detail` table is a partitioned table, whereas the table created by using the `create table...as select...` statement does not copy the partition attribute. The partition key column of the source table becomes a common column in the object table. This means that `sale_detail_ctas1` is a non-partitioned table with five columns.

In the `create table...as select...` statement, if you use constants as column values in the `SELECT` clause, we recommend that you specify the column name. Example of the statement:

```
CREATE TABLE sale_detail_ctas2
AS
SELECT shop_name, customer_id, total_price, '2013' AS sale_date, 'China' AS region
FROM sale_detail;
```

If the column name is not specified, the following statement is used:

```
CREATE TABLE sale_detail_ctas3
AS
SELECT shop_name, customer_id, total_price, '2013', 'China'
FROM sale_detail;
```

Then, the fourth column and fifth column of the created table `sale_detail_ctas3` become system generated names, like `_c5` and `_c6`.

To allow the destination table to have the same structure as the source table, try to use the following `create table...like` statement:

```
create table sale_detail_like like sale_detail;
```

Now the table structure of `sale_detail_like` is exactly the same as `sale_detail`. Except for the lifecycle, both tables have the same column names, column comments, and table comment. However, data in the `sale_detail` table is not duplicated to the `sale_detail_like` table.

To create a hash cluster table, you can use the following statements:


```
CREATE TABLE T1 (a string, b string, c bigint) CLUSTERED BY (c) SORTED BY (c) INTO 1024 BUCKETS; -- Creates a hash cluster non-partitioned table.
```

```
CREATE TABLE T1 (a string, b string, c bigint) PARTITIONED BY (dt string) CLUSTERED BY (c) SORTED BY (c) INTO 1024 BUCKETS; -- Creates a hash cluster partitioned table.
```

View table information

Syntax:

```
desc <table_name>;  
desc extended <table_name>; -- View external table information.
```

Examples

- To view the information of the preceding table `sale_detail`, execute the following statement:

```
desc sale_detail;
```

The following result is returned:

```

odps@ $odps_project>desc sale_detail;
+-----+
| Owner: ALIYUN$maoXXX@alibaba-inc.com | Project: $odps_project
|
| TableComment:
|
+-----+
| CreateTime:      2017-06-28 15:05:17
|
| LastDDLTime:     2017-06-28 15:05:17
|
| LastModifiedTime: 2017-06-28 15:05:17
|
+-----+
| InternalTable: YES | Size: 0
|
+-----+
| Native Columns:
|
+-----+
| Field    | Type   | Label | Comment
|
+-----+
| shop_name | string |      |
|
| customer_id | string |      |
|
| total_price | double |      |
|
+-----+
| Partition Columns:
|
+-----+
| sale_date | string |
|
| region    | string |
|
+-----+
OK

```

- To view the information of the preceding table `sale_detail_like` , execute the following statement:

```
desc sale_detail_like;
```

The following result is returned:

```
odps@ $odps_project>desc sale_detail_like;
+-----+
| Owner: ALIYUN$maoXXX@alibaba-inc.com | Project: $odps_project
|
| TableComment:
|
+-----+
| CreateTime:      2017-06-28 15:42:17
|
| LastDDLTime:     2017-06-28 15:42:17
|
| LastModifiedTime: 2017-06-28 15:42:17
|
+-----+
| InternalTable: YES | Size: 0
|
+-----+
| Native Columns:
|
+-----+
| Field | Type | Label | Comment
|
+-----+
| shop_name | string | |
|
| customer_id | string | |
|
| total_price | double | |
|
+-----+
| Partition Columns:
|
+-----+
| sale_date | string |
|
| region | string |
|
+-----+
OK
```

In the preceding examples, the attributes of `sale_detail_like` are the same as those of `sale_detail`, except for the lifecycle. The same attributes include the field type and partition type. For more information, see [Table-level operations](#).

Note The data size you obtain by using the `describe table` command includes the size of data in the recycle bin. If you want to clear the recycle bin, run the `purge table table_name` statement. Then, run the `describe table` statement to view the size of data that excludes the size of data in the recycle bin. You can also run the `show recyclebin` statement to view the details about data in the recycle bin for the current project.

If you view the information about the `sale_detail_ctas1` table, you can find that `sale_date` and `region` are only general columns and not partitions of the table.

The types of data that the `DESC` statement can return increase with the data types supported by MaxCompute. For more information, see [Date types](#). If new data types are used in MaxCompute, you need to enable new data types when you execute SQL statements. However, you do not need to do so when you run the `desc` command.

Note If the output of SQL statements depends on the input of the `desc table` command, we recommend that you promptly update settings to parse new data types in MaxCompute.

Examples

```
set odps.sql.type.system.odps2=true;
CREATE TABLE test_newtype (
  c1 tinyint
,c2 smallint
,c3 int
,c4 BIGINT
,c5 float
,c6 DOUBLE
,c7 decimal
,c8 binary
,c9 timestamp
,c10 ARRAY<map<BIGINT,BIGINT>>
,c11 map<STRING,ARRAY<BIGINT>>
,c12 STRUCT<s1:STRING,s2:BIGINT>
,c13 varchar(20))
LIFECYCLE 1
;
```

The following result is returned by the `desc test_newtype;` statement, with some information being omitted:

Native Columns:				
+-----+				
Field	Type	Label	Comment	
+-----+				
c1	tinyint			
c2	smallint			
c3	int			
c4	bigint			
c5	float			
c6	double			
c7	decimal			
c8	binary			
c9	timestamp			
c10	array<map<bigint,bigint>>			
c11	map<string,array<bigint>>			
c12	struct<s1:string,s2:bigint>			
c13	varchar(20)			
+-----+				

You can run the `DESC EXTENDED table_name;` statement to view the Clustering attribute of a hash cluster table. In the following result, the Clustering attribute is displayed in Extended Info.

```

+-----+
| Owner: ALIYUN$xxxxxxx@aliyun.com | Project: xxxxx |
| TableComment: |
+-----+
| CreateTime: 2017-12-25 11:18:26 |
| LastDDLTime: 2017-12-25 11:18:26 |
| LastModifiedTime: 2017-12-25 11:18:26 |
| Lifecycle: 2 |
+-----+
| InternalTable: YES | Size: 0 |
+-----+
| Native Columns: |
+-----+
| Field | Type | Label | Comment |
+-----+
| a | string | | |
| b | string | | |
| c | bigint | | |
+-----+
| Partition Columns: |
+-----+
| dt | string | |
+-----+
| Extended Info: |
+-----+
| TableID: 91a3395d3ef64b4d9ee1d2852755 |
| IsArchived: false |
| PhysicalSize: 0 |
| FileNum: 0 |
| ClusterType: hash |
| BucketNum: 1024 |
| ClusterColumns: [c] |
| SortColumns: [c ASC] |
+-----+

```

If the table is a partitioned table that has the Clustering attribute, run the following command to view the partition attributes in addition to running the preceding command to view the table attributes:

```
DESC EXTENDED table_name partition(pt_spec);
```

Run the `SELECT` statement to view the details of the table. For more information, see [SELECT syntax](#).

View table creation statements

Syntax:

```
SHOW CREATE TABLE <table_name>;
```

 **Note** You can run the `SHOW CREATE TABLE` statement to generate SQL DDL statements that are used to create a table and use the statements to rebuild the table schema.

Drop a table

Syntax:

```
DROP TABLE [IF EXISTS] table_name;
```

-  **Note**
- If you do not specify the `IF EXISTS` option and the target table does not exist, an error is returned. If you specify this option, a success message is returned regardless of whether the table exists.
 - If you delete an external table, data in Object Storage Service (OSS) is not deleted.

Example:

```
create table sale_detail_drop like sale_detail;
drop table sale_detail_drop;
-- If the table exists, a success message is returned. If the table does not exist, an error is returned.
drop table if exists sale_detail_drop2;
-- A success message is returned regardless of whether the sale_detail_drop2 table exists.
```

Rename a table

Syntax:

```
ALTER TABLE table_name RENAME TO new_table_name;
```


 Note

- The preceding statement is used to rename a table. The `RENAME` operation only changes the table name. The table data is not changed.
- If the table specified by `new_table_name` already exists, an error is returned.
- If the table specified by `table_name` does not exist, an error is returned.

Example:

```
create table sale_detail_rename1 like sale_detail;  
alter table sale_detail_rename1 rename to sale_detail_rename2;
```

Change the owner of a table

MaxCompute SQL allows you to execute the `CHANGEOWNER` statement to change the owner of a table. Syntax:

```
alter table table_name changeowner to 'ALIYUN$xxx@aliyun.com';
```

Change the comments of a table

Syntax:

```
ALTER TABLE table_name SET COMMENT 'tbl comment';
```

 Note

- The table specified by `table_name` must exist.
- The comment length must not exceed 1024 bytes.

Example:

```
alter table sale_detail set comment 'new coments for table sale_detail';
```

You can run the `DESC` command in MaxCompute to view the modified `comment` of the table. For more information, see `Describe Table` in [Table operations](#).

Change the last update time of a table

MaxCompute SQL allows you to run the `TOUCH` statement to change the value of `LastDataModifiedTime`. The value can be changed to the current time.

Syntax:

```
ALTER TABLE table_name TOUCH;
```

 Note

- If the table specified by `table_name` does not exist, an error is returned.
- After the value of `LastDataModifiedTime` is changed by using this statement, MaxCompute considers that the table data has changed, and restarts the lifecycle of the table from the last update time.

Change the Hash Clustering attribute of a table

You can run the `ALTER TABLE` statement to add the Hash Clustering attribute to a partitioned table or remove the Hash Clustering attribute from a partitioned table.

Syntax:

```
ALTER TABLE table_name
[CLUSTERED BY (col_name [, col_name, ...]) [SORTED BY (col_name [ASC | DESC] [, col_name [ASC | DESC] .
..))] INTO number_of_buckets BUCKETS]
```

The following statement is used to remove the Hash Clustering attribute from a table:

```
ALTER TABLE table_name NOT CLUSTERED;
```

 Note

- The `ALTER TABLE` statement can only modify the Hash Clustering attribute for a partitioned table instead of a non-partitioned table. The Hash Clustering attribute cannot be modified after it is added for a non-partitioned table.
- The `ALTER TABLE` statement takes effect only on the new partitions of a table. Therefore, you do not need to execute the `PARTITION ALTER TABLE` statement for the existing partitions. After the Hash Clustering attribute is added, data is stored to the new partitions in compliance with hash clustering.

Clear data from a non-partitioned table

You can use the `TRUNCATE TABLE` statement to clear data from a specified non-partitioned table instead of a partitioned table. For a partitioned table, you can run the `ALTER TABLE table_name DROP PARTITION` statement to clear data from partitions.

Syntax:

```
TRUNCATE TABLE table_name;
```

3.5.2. Lifecycle management operations

MaxCompute provides the lifecycle management function to facilitate the release of storage space and simplify data reclamation.

lifecycle

You can specify the lifecycle when you create a table. The lifecycle can only be set for a table rather than a partition. For more information, see [Table-level operations](#).

- In MaxCompute, once data in a table is modified, the `LastDataModifiedTime` value is updated. Therefore, MaxCompute determines whether to reclaim this table based on the `LastDataModifiedTime` value and lifecycle.
- For a non-partitioned table, if the data remains unchanged for `days` after the last data modification, MaxCompute automatically executes a statement such as `DROP TABLE` to reclaim the table.
- For a partitioned table, MaxCompute determines whether to reclaim a partition based on its `LastDataModifiedTime` value.
- Unlike non-partitioned tables, a partitioned table is not deleted after the last partition is reclaimed.
- Instead of deleting the lifecycle configured for a non-partitioned table, you can only modify the lifecycle setting. The lifecycle configured for a specific partition in a partitioned table can be deleted.

Change the lifecycle configured for a table

Syntax

```
ALTER TABLE table_name SET lifecycle days;
```

Description: used to change the lifecycle configured for a table.

Parameters

- `table_name`: the name of the table whose lifecycle setting you want to modify.
- `days`: the lifecycle duration, which must be a positive integer in days.

Example

```
-- Create a table named test_lifecycle with a lifecycle of 100 days.  
CREATE TABLE test_lifecycle(key string) lifecycle 100;  
-- Change the lifecycle of test_lifecycle to 50 days.  
ALTER TABLE test_lifecycle SET lifecycle 50;
```

Disable or restore the lifecycle feature

Syntax

```
ALTER TABLE table_name [partition_spec] ENABLE|DISABLE LIFECYCLE;
```

Description: used to enable or disable the lifecycle feature for a table or a specific partition in a table.

Parameters

- **table_name**: the name of the table.
- **partition_spec**: the name of the partition.
- **DISABLE LIFECYCLE**: used to disable the lifecycle feature for the table or partition.
 - It prevents the reclamation of a table and its partitions based on the lifecycle feature. This parameter has a higher priority than `partition_spec enable lifecycle`.
 - After the lifecycle feature of a table is disabled, the table lifecycle setting and the **ENABLE** and **DISABLE** tags of partitions in the table are retained.
 - After the lifecycle feature of a table is disabled, you can still modify the lifecycle settings of the table and its partitions.
- **ENABLE LIFECYCLE**: used to enable the lifecycle feature for the table or partition.
 - After the lifecycle feature is enabled again, a table and its partitions can be reclaimed based on the lifecycle feature. By default, the lifecycle settings of the current table and its partitions are used.
 - Before you enable the lifecycle feature for a table, you can modify the lifecycle settings of the table and its partitions as needed. This prevents data from being mistakenly reclaimed due to the use of previous settings.

Example

```
-- Disable the lifecycle feature for table trans.  
ALTER TABLE trans DISABLE LIFECYCLE;  
  
-- Disable the lifecycle feature for the dt='20141111' partition in table trans.  
ALTER TABLE trans PARTITION(dt='20141111') DISABLE LIFECYCLE;
```

3.5.3. Partition and column operations

This topic describes how to execute statements to add or delete partitions and add or modify columns in MaxCompute tables.

Add a partition

Syntax

```
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec;  
partition_spec:(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)
```

Description

This statement is used to add a partition to an existing table. This statement can add only a partition instead of a partition field.

 Note

- You can create a maximum of 60,000 partitions in a single table in MaxCompute.
- To add a partition to a table that has multi-level partitions, you must specify all partition column values.

Parameters

- **table_name**: the name of the destination table to which you want to add a partition.
- **IF NOT EXISTS**: If IF NOT EXISTS is not specified and a partition with an identical name already exists, an error is returned.
- **partition_spec**: the name of the new partition. Partition names must be in lowercase.

Examples

```
-- Add a partition to the sale_detail table to store the sales records of Hangzhou in December 2013.
alter table sale_detail add if not exists partition (sale_date='201312', region='hangzhou');

-- Add a partition to the sale_detail table to store the sales records of Shanghai in December 2013.
alter table sale_detail add if not exists partition (sale_date='201312', region='shanghai');

-- Add a partition to the sale_detail table to store the sales records on October 11, 2011. The addition fails and an error is returned.
alter table sale_detail add if not exists partition(sale_date='20111011');

-- Add a partition to the sale_detail table to store the sales records of Shanghai. The addition fails and an error is returned.
alter table sale_detail add if not exists partition(region='shanghai');
```

Drop a partition

Syntax

```
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec;
partition_spec:(partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)
```

Parameters

- **table_name**: the name of the destination table to which you want to drop a partition.
- **IF EXISTS**: If IF EXISTS is not specified and the partition does not exist, an error is returned.
- **partition_spec**: the name of the partition you want to delete. Partition names must be in lowercase.

Example

```
-- Delete the sales records of Hangzhou in December 2013 from the sale_detail table.  
alter table sale_detail drop if exists partition(sale_date='201312',region='hangzhou');
```

Add columns

Syntax

- Add columns to a table

```
ALTER TABLE table_name ADD COLUMNS (col_name1 type1,col_name2 type2...);
```

- Add columns and comments at the same time

```
ALTER TABLE table_name ADD COLUMNS (col_name1 type1 comment 'XXX',col_name2 type2 comment 'XXX');
```

Parameters

- **table_name**: the name of the destination table to which you want to add a column. You cannot specify the order of a new column in the table. The new column is added as the last column by default.
- **col_name, type, comment**: the name, data type, and comment of the column you want to add.

Modify column name

Syntax

```
ALTER TABLE table_name CHANGE COLUMN old_col_name RENAME TO new_col_name;
```

Parameters

- **table_name**: the name of the table whose column name you want to modify.
- **old_col_name**: the name of the column you want to modify. The column specified by `old_col_name` must already exist in the table.
- **new_col_name**: the name of the new column. The table does not contain a column named `new_col_name`.

Modify the comment of a column

Syntax

```
ALTER TABLE table_name CHANGE COLUMN col_name COMMENT comment_string;
```

Parameters

- **table_name**: the name of the table for which you want to modify the column comment.
- **col_name**: the name of the column for which you want to modify comment. The column specified by `col_name` must already exist in the table.
- **comment_string**: the information about the new comment. The maximum size of a comment is 1024 bytes.

Modify column names and column comments simultaneously

Syntax

```
ALTER TABLE table_name CHANGE COLUMN old_col_name new_col_name column_type COMMENT column_comment;
```

Parameters

- **table_name**: the name of the table for which you want to modify the column names and comments.
- **old_col_name**: the name of the column for which you want to modify the comment. The column specified by **old_col_name** must already exist in the table.
- **new_col_name**: the new column name. The table does not contain a column named **new_col_name**.
- **column_type**: the data type of the column.
- **column_comment**: the information about the new comment. The maximum size of a comment is 1024 bytes.

Modify LastDataModifiedTime of a table or partition

MaxCompute SQL allows you to execute the **TOUCH** statement to modify

LastDataModifiedTime of a partition. This statement modifies **LastDataModifiedTime** of a partition to the current time. In this case, MaxCompute considers that the table and partition data have changed, and the new lifecycle of the table starts from the last update time.

Syntax

```
ALTER TABLE table_name TOUCH PARTITION(partition_col='partition_col_value', ...) ;
```

Parameters

- **table_name**: the name of the table whose **LastDataModifiedTime** you want to modify. If the table does not exist, an error is returned.
- **partition_col='partition_col_value'**: the name and column value of the partition whose **LastDataModifiedTime** you want to modify. If the specified partition and its column value do not exist, an error is returned.

Modify the partition column value

MaxCompute SQL allows you to execute the **RENAME** statement to change the partition column values of a table.

Syntax

```
ALTER TABLE table_name PARTITION (partition_col1 = partition_col_value1, partition_col2 = partition_col_value2, ...)
RENAME TO PARTITION (partition_col1 = partition_col_newvalue1, partition_col2 = partition_col_newvalue2, ...) ;
```

Parameters

- **table_name**: the name of the table whose partition column value you want to change. If the table does not exist, an error is returned.
- **partition_col = partition_col_value**: the name and column value of the partition in a table.
- **partition_col = partition_col_newvalue**: the name and new column value of the partition in a table.

Note

- This statement can only change partition column values instead of partition column names.
- To change one or more partition column values in a table that has multi-level partitions, you must specify the partition column values of partitions at each level.

Merge partitions

MaxCompute SQL allows you to execute the **MERGE PARTITION** statement to merge multiple partitions of a table into one partition. This statement deletes the dimension information about the merged partitions and transfers data to the specified partition.

Syntax

```
ALTER TABLE <tableName> MERGE [IF EXISTS] PARTITION(<predicate>) [, PARTITION(<predicate2>) ...]
OVERWRITE PARTITION(<fullPartitionSpec>) [PURGE];
```

Syntax description

- If the partition does not exist and **IF EXISTS** is not specified, an error is returned.
- If no partitions meet merge conditions after **IF EXISTS** is specified, a new partition is not generated.
- If the source data is modified by executing the **INSERT**, **RENAME**, and **DROP** statements concurrently, an error is returned even if **IF EXISTS** is specified.
- You cannot merge partitions of an external table or a table shard. If you merge partitions of a clustered table, the clustered attribute of the table is removed.
- You can merge a maximum of 4,000 partitions at a time.

Example

```
-- Partitions and data of the table:
odps@jet_zwz>list partitions intpstringstringstring;

ds=20181101/hh=00/mm=00
ds=20181101/hh=00/mm=10
ds=20181101/hh=10/mm=00
ds=20181101/hh=10/mm=10

OK
odps@jet_zwz>read intostringstringstring;
```



```
odps@ jet_zwz>read intpstringstringstring;
```

```
+-----+-----+-----+-----+
| value  | ds      | hh    | mm    |
+-----+-----+-----+-----+
| 1      | 20181101 | 00    | 00    |
| 1      | 20181101 | 00    | 10    |
| 1      | 20181101 | 10    | 00    |
| 1      | 20181101 | 10    | 10    |
+-----+-----+-----+-----+
```

-- Merge all partitions that meet the hh='00' condition into the ds='20181101', hh='00', mm='00' partition:

```
odps@ jet_zwz>alter table intpstringstringstring merge partition(hh='00') overwrite partition(ds='20181101', hh='00', mm='00');
```

```
ID = 20190404025755844g80qwa7a
```

```
OK
```

-- View the merged partitions.

```
odps@ jet_zwz>list partitions intpstringstringstring;
```

```
ds=20181101/hh=00/mm=00
```

```
ds=20181101/hh=10/mm=00
```

```
ds=20181101/hh=10/mm=10
```

```
OK
```

-- Data in two partitions that meet the hh='00' condition is merged into the ds='20181101', hh='00', mm='00' partition.

```
odps@ jet_zwz>read intpstringstringstring;
```

```
+-----+-----+-----+-----+
| value  | ds      | hh    | mm    |
+-----+-----+-----+-----+
| 1      | 20181101 | 00    | 00    |
| 1      | 20181101 | 00    | 00    |
| 1      | 20181101 | 10    | 00    |
| 1      | 20181101 | 10    | 10    |
+-----+-----+-----+-----+
```

When you merge partitions, you can specify multiple predicate conditions. For example, you can execute the following statement to merge all the remaining partitions to the `ds=20181101/hh=00/mm=00` partition:

```
odps@ jet_zwz>alter table intpstringstringstring merge if exists partition(ds='20181101', hh='00', mm='00'), partition(ds='20181101', hh='10', mm='00'), partition(ds='20181101', hh='10', mm='10') overwrite partition(ds='20181101', hh='00', mm='00') purge;
```

```
ID = 20190404034632854g431sqzt2
```

```
OK
```

```
odps@ jet_zwz>show partitions intpstringstringstring;
```

```
ds=20181101/hh=00/mm=00
```

```
OK
```

3.5.4. View-related operations

This topic describes how to use DDL statements to create, delete, and rename a view.

create a view delete a view rename a view

Create a view

Syntax

```
CREATE [OR REPLACE] VIEW [IF NOT EXISTS] view_name  
  [(col_name [COMMENT col_comment], ...)]  
  [COMMENT view_comment]  
  [AS select_statement]
```

Description

This syntax is used to create a view.

**Notice**

- To create a view, you must have read permissions on the table referenced by the view.
- A view can reference other views but cannot reference itself. Circular reference is not supported.
- Writing data into a view is not allowed. For example, you cannot manage a view by using the `INSERT INTO` or `INSERT OVERWRITE` statement.
- After a view is created, it may be inaccessible if the referenced table is altered. For example, a referenced table is deleted. You must maintain the mappings between referenced tables and views.

Parameters

- `view_name`: the name of the created view.
- `IF NOT EXISTS`: If `IF NOT EXISTS` is not specified and the view already exists, the execution of the `CREATE VIEW` statement causes exceptions. In this case, you can execute `CREATE OR REPLACE VIEW` to recreate the view. The permissions on the view remain unchanged after the view is recreated.
- `col_name`: the name of the column contained in the view.
- `view_comment`: the comment of the view.
- `select_statement`: the query statement, which is the data source of the view. A view can contain only one valid `SELECT` statement.

Example

```
-- Create a view named sale_detail_view.  
create view if not exists sale_detail_view  
(store_name, customer_id, price, sale_date, region)  
comment 'a view for table sale_detail'  
as select * from sale_detail;
```

Rename a view**Syntax**

```
ALTER VIEW view_name RENAME TO new_view_name;
```

Parameters

- `view_name`: the name of the view you want to rename.
- `new_view_name`: the new name of the view. If a view with the same name already exists, an error is returned.

Example

```
-- Rename sale_detail_view to market.  
alter view sale_detail_view rename to market;
```

Delete a view

Syntax

```
DROP VIEW [IF EXISTS] view_name;
```

Parameters

- **view_name**: the name of the view you want to delete.
- **IF NOT EXISTS**: If the view does not exist and **IF NOT EXISTS** is not specified, an error is returned.

Example

```
-- Delete sale_detail_view.  
DROP VIEW IF EXISTS sale_detail_view;
```

3.6. Insert Operation

3.6.1. MULTI INSERT

MaxCompute SQL supports inserting different result tables or partitions in a single SQL statement.

Statement format:

```
FROM from_statement  
INSERT OVERWRITE | INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)]  
select_statement1 [FROM from_statement]  
[INSERT OVERWRITE | INTO TABLE tablename2 [PARTITION (partcol1=val3, partcol2=val4 ...)]  
select_statement2 [FROM from_statement]]
```

 Note

- Generally, up to 255 ways of output can be written in a single SQL statement. A syntax error occurs, if the output exceeds 255 ways.
- In a multi insert statement:
 - For a partitioned table, a target partition cannot appear multiple times.
 - For an unpartitioned table, this table cannot appear multiple times.
- Different partitions within a partitioned table cannot have an Insert overwrite operation and an Insert into operation at the same time; otherwise, an error is returned.

For an unpartitioned table, this table cannot appear multiple times.

```
create table sale_detail_multi like sale_detail;
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price where .....
insert overwrite table sale_detail_multi partition (sale_date='2011', region='china' )
select shop_name, customer_id, total_price where .....
-- Return result successfully. Insert the data of sale_detail into the 2010 sales records and 2011 sales records in China region.
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price;
-- An error is thrown. The same partition appears for multiple times.
from sale_detail
insert overwrite table sale_detail_multi partition (sale_date='2010', region='china' )
select shop_name, customer_id, total_price
insert into table sale_detail_multi partition (sale_date='2011', region='china' )
select shop_name, customer_id, total_price;
-- An error is thrown. Different partitions within a partition table cannot have both an 'insert overwrite' operation and an 'insert into' operation.
```

3.6.2. Insert data in dynamic partition mode (DYNAMIC PARTITION)

To insert data into a partitioned table by using the **INSERT OVERWRITE** statement, you can specify a partitioning column value in the statement. You can also specify the name of a partitioning column in the **INSERT OVERWRITE** statement but set the partitioning column value in the **SELECT** clause. The method of specifying the name of a partitioning column in the **INSERT OVERWRITE** statement indicates that data is inserted in dynamic partition mode. This topic describes the basic syntax and provides examples on how to insert data in dynamic partition mode.

Syntax

Statement

```
INSERT OVERWRITE TABLE tablename PARTITION (partcol1, partcol2 ...) select_statement FROM from_statement;
```

Parameters

- **tablename**

The name of the destination table into which you want to insert the data.

- **partcol1, partcol2 ...**

The names of partitioning columns in the destination table into which you want to insert the data.

- **select_statement**

The statement used to query data in the source table. The fields in the **select_statement** clause provide the dynamic partition values in the destination table. If the destination table only has level-1 dynamic partitions, the value of the last field in the **select_statement** clause is the dynamic partition value in the destination table.

Note

- If you execute an SQL statement that supports the dynamic partition feature in a distributed environment, a single process can provide up to 512 dynamic partitions. If the number of dynamic partitions exceeds this limit, an exception occurs.
- An SQL statement that supports the dynamic partition feature can generate up to 2,000 dynamic partitions. If the number of dynamic partitions exceeds this limit, an exception occurs.
- A dynamic partition value cannot be NULL or contain special characters. Otherwise, **FAILED: ODPS-0123031:Partition exception - invalid dynamic partition value: province=xxx** is returned.
- If a destination table has multi-level partitions, you can specify some partitions as static partitions in an **INSERT** statement. However, the static partitions must be high-level partitions.
- If a destination table is a hash clustering table, the dynamic partition feature is not supported.

Precautions

In dynamic partition mode, the mapping between the fields in the `select_statement` clause and dynamic partitions in a destination table is determined by the field order, not by the column name. If the field order in a source table is different from that in a destination table, we recommend that you specify the fields in the `select_statement` clause based on the order in the destination table.

For example, create destination table `sale_detail_dypart`.

```
create table sale_detail_dypart like sale_detail;
```

- Execute the following statements to insert the data from source table `sale_detail` into destination table `sale_detail_dypart`:

```
insert overwrite table sale_detail_dypart partition (sale_date, region)
select shop_name, customer_id, total_price, sale_date, region from sale_detail;
```

In these statements, `sale_date` in `sale_detail` determines the value of the `sale_date` dynamic partition in `sale_detail_dypart`, and `region` in `sale_detail` determines the value of the `region` dynamic partition in `sale_detail_dypart`.

- Execute the following statements to insert the data from `sale_detail` into `sale_detail_dypart`:

```
insert overwrite table sale_detail_dypart partition (sale_date, region)
select shop_name, customer_id, total_price, region, sale_date from sale_detail;
```

In these statements, `region` in `sale_detail` determines the value of the `sale_date` dynamic partition in `sale_detail_dypart`, and `sale_date` in `sale_detail` determines the value of the `region` dynamic partition in `sale_detail_dypart`.

 **Note** By default, the full scan is disabled for partitioned tables of a newly created project. If a full table scan statement is executed on a partitioned table of the project, a failure message is returned. To address this issue, you must specify a partition condition during the query. Therefore, when you execute a full table scan statement, you must insert `set odps.sql.allow.fullscan=true;` before the statement and then commit the statements for execution.

Examples

- Example 1

```
-- Create destination table total_revenues.
create table total_revenues (revenue bigint) partitioned by (region string);

-- Insert the data from sale_detail to total_revenues.
insert overwrite table total_revenues partition(region)
select total_price as revenue, region from sale_detail;
```

You can obtain the partitions generated based on the `region` field only after the `SELECT` statement is executed.

- Example 2: Insert the data from a source table into a destination table. If the destination table has multi-level partitions, level-1 partition `sale_date` must be specified.

```
insert overwrite table sale_detail_dypart partition (sale_date='2013', region)
select shop_name,customer_id,total_price,region from sale_detail;
```

- Example 3: To insert data in dynamic partition mode, the dynamic partition column must be included in the `SELECT` statement. Otherwise, the execution fails.

```
insert overwrite table sale_detail_dypart partition (sale_date='2013', region)
select shop_name,customer_id,total_price from sale_detail;
```

- Example 4: If you specify only low-level sub-partitions when you insert data in dynamic partition mode, inserting data into high-level partitions will fail.

```
insert overwrite table sales partition (region='china', sale_date)
select shop_name,customer_id,total_price,sale_date from sale_detail;
```

- Example 5: If the types of values in partitioning columns are inconsistent with those in the `SELECT` clause, an error is returned when data is inserted in dynamic partition mode in MaxCompute V1.0. MaxCompute V2.0 supports the **implicit conversion of data types**.

```
-- Create destination table parttable.
create table parttable(a int, b double) partitioned by (p string);
-- Insert the data from source table src into parttable.
insert into parttable partition(p) select key, value, current_timestmap() from src;
-- Query data in parttable.
select * from parttable;
```

The following table lists the results.

a	b	p
0	NULL	2017-01-23 22:30:47.130406621
0	NULL	2017-01-23 22:30:47.130406621

 **Note** If your data is ordered, it is randomly scattered when it is inserted in dynamic partition mode, which reduces the compression ratio. In this case, we recommend that you use **Tunnel commands** to upload the data to dynamic partitions to increase the compression ratio. For more information, see **Migrate data from RDS to MaxCompute to implement dynamic partitioning**.

3.6.3. VALUES

This topic describes the `INSERT ... VALUES` command operation.

In the test phase, you need to prepare some data for simple testing:

- If you want to create a few data records (such as 10 or fewer records), you can use the `INSERT ... VALUES` statement to quickly write the data.
- If you want to create a large number of data records (such as more than 10 records), you can incorporate the data records into a .txt or .csv file and then upload the file by using Tunnel. For more information about this method, see [Import data](#). Alternatively, you can incorporate the data records into a file and then import the file by using DataWorks.

 Note

Currently, the `INSERT OVERWRITE` statement cannot insert columns, therefore we recommend that you use the `INSERT INTO` statement instead.

Statement format:

```
INSERT INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)][colname1,colname2...]
[VALUES (col1_value,col2_value,...),(col1_value,col2_value,...),...]
```

Example 1::

```
drop table if exists srcp;
create table if not exists srcp (key string ,value bigint) partitioned by (p string);
insert into table srcp partition (p='abc') values ('a',1),('b',2),('c',3);
```

After the preceding statements are complete, the result of Partition `abc` is as follows:

```
| key | value | p |
| a | 1 | abc |
| b | 2 | abc |
| c | 3 | abc |
```

If you want to run the `INSERT` statement to write data into only some columns of a table, use the `insert list` function as follows.

Example 2:

```
drop table if exists srcp;
create table if not exists srcp (key string ,value bigint) partitioned by (p string);
insert into table srcp partition (p)(key,p) values ('d','20170101'),('e','20170101'),('f','20170101');
```

After the preceding statements are complete, the result of Partition 20170101 is as follows:

```
| key | value | p |
| d | NULL | 20170101 |
| e | NULL | 20170101 |
| f | NULL | 20170101 |
```

For columns not specified in values, the default value is *NULL*. The insert list function is not necessarily used with values, and can also be used with the insert into...select... function.

In fact, the values is not only used in the INSERT statement, any DML statement can also be used.

The INSERT...VALUES method has a limitation: values must be constants. You can use the values table function of MaxCompute to perform some simple operations on the inserted data. For more information, see Example 3.

Example 3:

```
drop table if exists srcp;
create table if not exists srcp (key string ,value bigint) partitioned by (p string);
insert into table srcp partition (p) select concat(a,b), length(a)+length(b),'20170102' from values ('d',4),
('e',5),('f',6) t(a,b);
```

The values (...), (...) t(a, b) is used to define a table named t whose columns are a and b and data types are STRING for column a and BIGINT for column b. The data types are derived from the values list. In this way, with no physical table prepared, it is possible to simulate a multi-row table with arbitrary data and perform arbitrary calculations.

After the preceding statements are complete, the result of Partition 20170102 is as follows:

```
| key | value | p |  
  
| d4 | 2 | 20170102 |  
| e5 | 2 | 20170102 |  
| f6 | 2 | 20170102 |
```

The use of `VALUES TABLE` can also replace the combination of `select * from dual` and to spell out the constants as follows:

```
select 1 c from dual  
union all  
select 2 c from dual;  
--The same as:  
select * from values (1), (2) as t (c);
```

A special usage of values is as follows.

```
select abs(-1), length('abc'), getdate();
```

As the preceding statement shows, the `SELECT` statement can be run without the `FROM` statement, if the expression list of the `SELECT` statement does not use any upstream table data. The underlying implementation is selecting from an anonymous values table in one row and zero columns. In this way, to test some functions, such as your UD, you do not need to manually create DUAL tables.

 Note

- Values only support constants and do not support functions including ARRAY complex types. Currently, MaxCompute cannot construct corresponding constants. Therefore, we recommend that you modify the statement as follows:

```
insert into table srcp (p ='abc') select 'a',array('1', '2', '3');
```

which can provide the same effect.

- To write the DATETIME and TIMESTAMP data types through values, specify the type names in the **VALUES** statement, for example:

```
insert into table srcp (p ='abc') values (datetime'2017-11-11
00:00:00',timestamp'2017-11-11 00:00:00.123456789');
```

3.7. Lateral View

Lateral view is used in conjunction with UDTF such as split, explode, etc. It can split a row of data into multiple rows, and aggregate the split data on this basis.

Single Lateral View statement

Syntax:

```
lateralView: LATERAL VIEW [OUTER] udtf(expression) tableAlias AS columnAlias (' columnAlias) * from
Clause: FROM baseTable (lateralView)*
```

Notes:

- Lateral view outer: When the table function does not output any rows, the corresponding input rows remain in the Lateral View results, and all table function output lists are null.

Example:

Suppose we have a table called "pageAds" which has two columns of data. The first column is "pageid string" and the second column is "adid_list", a comma-separated collection of AD IDs.

string pageid	Array<int> adid_list
"front_page"	[1, 2, 3]
"contact_page"	[3, 4, 5]

The requirement is to count the number of times all AD IDs have appeared. The implementation process is as follows.

1. Split the AD IDs as follows:

```
SELECT pageid, adid
FROM pageAds LATERAL VIEW explode(adid_list) adTable AS adid;
```

The execution result is as follows:

string pageid	int adid
"front_page"	1
"front_page"	2
"front_page"	3
"contact_page"	3
"contact_page"	4
"contact_page"	5

2. The statistics for the aggregation:

```
SELECT adid, count(1)
  FROM pageAds LATERAL VIEW explode(adid_list) adTable AS adid
 GROUP BY adid;
```

Result:

int adid	count(1)
1	1
2	1
3	2
4.	1
50	1

Multiple Lateral View statements

A from statement can be followed by multiple Lateral View statements, the subsequent Lateral View statement can reference all the former tables and columns.

The following table is an example:

Array<int> col1	Array<string> col2
[1, 2]	["a" , "b" , "c"]
[3, 4]	["d" , "e" , "f"]

- Execute a single statement:

```
SELECT myCol1, col2 FROM baseTable
  LATERAL VIEW explode(col1) myTable1 AS myCol1;
```

Result:

int mycol1	Array<string> col2
1	["a" , "b" , "c"]
2	["a" , "b" , "c"]
3	[d" , "e" , "f"]
4	[d" , "e" , "f"]

- Add a Lateral View statement as follows:

```
SELECT myCol1, myCol2 FROM baseTable
  LATERAL VIEW explode(col1) myTable1 AS myCol1
  LATERAL VIEW explode(col2) myTable2 AS myCol2;
```

Result is as follows:

int myCol1	string myCol2
1	"a"
1	"b"
1	"c"
2	"a"
2	"b"
2	"c"
3	"d"
3	"e"
3	"f"
4	"d"
4	"e"
4	"f"

3.8. Select Operation

3.8.1. SELECT syntax

This topic describes MaxCompute `SELECT` syntax and the precautions for executing nested queries, sorting operations, and grouping queries by using `SELECT` syntax.

SELECT statement format

The `SELECT` statement format is as follows:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...  
FROM table_reference  
[WHERE where_condition]  
[GROUP BY col_list]  
[ORDER BY order_condition]  
[DISTRIBUTE BY distribute_condition [SORT BY sort_condition] ]  
[LIMIT number]
```

Column expressions

When using the `SELECT` statement to read data from a table, specify the names of the columns to be read, or use an asterisk (*) wildcard to specify all columns. The following is an example `SELECT` statement:

```
select * from sale_detail;
```

To specify only the `shop_name` column in *the sale_detail table*, use the following statement:

```
select shop_name from sale_detail;
```

Use a `WHERE` clause when you specify a filtering condition. For example:

```
select * from sale_detail where shop_name like 'hang%';
```

When you use a `SELECT` statement, up to 10,000 rows of results can be displayed. If the `SELECT` statement serves as a clause, all the results are returned to the upper-level query.

Note:

- Full table scans cannot be specified when `SELECT` statements are used on a partitioned table.

For new projects created after January 10, 2018, full table scans cannot be specified for a partitioned table in a project. Instead, you must specify each partition condition to be scanned. This is to help reduce excessive resource usage and lower costs. If your instances are billed by the Pay-As-You-Go billing method, the total number of partition conditions to be scanned is regarded as one billable item.

If the table definition is `t1(c1,c2) partitioned by(ds)`, you cannot run the following statement in a new project. Otherwise, an error may occur:

```
Select * from t1 where c1=1;
Select * from t1 where (ds='20180202' or c2=3);
Select * from t1 left outer join t2 on a.id =b.id and a.ds=b.ds and b.ds='20180101');
--When Join statement is running, if the partition clipping condition is placed in where clause, the partition clipping takes effect. If you put it in on clause, the partition clipping of sub table takes effect, and the main table performs a full table scan.
```

If you perform a full table scan on a partitioned table, you can add a set statement `set odps.sql.allow.fullscan=true;` before the SQL statement that is used to scan the partitioned table. The set statement must be entered together with the SQL statement. In the case that the *sales_detail* table is a partitioned table, submit the following simple query statements at the same time for a full table scan:

```
set odps.sql.allow.fullscan=true;
select * from sale_detail;
```

- `table_reference` supports nested subqueries, for example:

```
select * from (select region from sale_detail) t where region = 'shanghai';
```

 **Note** When you use a `SELECT` statement, up to 10,000 rows of results can be displayed. If the `SELECT` statement serves as a clause, all the results are returned to the upper-level query.

Use DISTINCT to remove duplicates

DISTINCT : If duplicated data rows exist, you can use the Distinct option before the field to remove duplicates. In this case, only one value is returned. If you use the ALL option, or do not specify this option, all duplicated values in the fields are returned.

If you use the Distinct option, only one row of a record is returned, which is shown as follows:

```
select distinct region from sale_detail;
select distinct region, sale_date from sale_detail;
-- Performs the Distinct option on multiple columns. The Distinct option has an effect on Select column sets rather than a single column.
```

WHERE clause conditions

The filter conditions supported by `WHERE` clauses as shown in the following table:

Filter criteria	Description
> ,< , = , >= , <= , <>	Relational operators
like, rlike	The source and pattern parameters of like and rlike can only be of the String type.

Filter criteria	Description
in, not in	If a subquery is attached to the in or not in a condition, only the values of one column are returned for the subquery, and the returned values cannot exceed 1,000 entries.

You can specify a partition scope in the `WHERE` clause of a `SELECT` statement that is used to scan specified partitions of a table instead of a whole table, shown as follows:

```
SELECT sale_detail.* FROM sale_detail WHERE sale_detail.sale_date >= '2008' AND sale_detail.sale_date <= '2014';
```

The `WHERE` clause of MaxCompute SQL supports query by using the between-and condition. The preceding SQL statement can be rewritten as follows:

```
SELECT sale_detail.* FROM sale_detail WHERE sale_detail.sale_date BETWEEN '2008' AND '2014';
```

Queries using GROUP BY

`GROUP BY` : Query by group. In most cases, `GROUP BY` is used in combination with an **aggregation function**. For a `SELECT` statement that contains an aggregate function to be used in combination with `GROUP BY`, it must comply with the following rules:

- Rule i: The key using `GROUP BY` is the name of a column in the input table.
- Rule ii: The key using `GROUP BY` is an expression consisting of columns of the input table. However, this expression cannot be the alias of an output column of the `SELECT` statement.

 **Note** In the case that rules i and ii are in conflicting with each other, rule i takes priority over rule ii. Specifically, if the key using `GROUP BY` is a column or expression of the input table and an output column of `SELECT`, rule i takes priority over rule ii.

For example:

```

select region from sale_detail group by region;
-- Runs successfully with the name of a column in the input table directly used as the group by column
select sum(total_price) from sale_detail group by region;
-- Runs successfully with the table grouped by the region value and returns the total sales of each group
up
Select region, sum (total_price) from sale_detail group by region;
-- Runs successfully with the table grouped by the region value and returns the region value (unique in the group) and total sales of each group
select region as r from sale_detail group by r;
-- Runs with the alias of the Select column and returns an error
select 2 + total_price as r from sale_detail group by 2 + total_price;
-- Requires a complete expression of the column
Select region, total_price from sale_detail group by region;
-- Returns an error; all columns not using an aggregate function in the Select statement must exist in group by
select region, total_price from sale_detail group by region, total_price;
-- Runs successfully

```

The preceding restrictions are imposed because **GROUP BY** operations come before **SELECT** operations during SQL parsing. Therefore, **GROUP BY** statements can only accept the columns or expressions of the input table as keys.

 **Note** For more information, see [Aggregate Functions](#).

Queries using ORDER BY, SORT BY, and DISTRIBUTE

- **ORDER BY** : Globally sorts all data based on certain columns. To sort records in descending order, use the DESC keyword. For global sorting, order by must be used together with limit. When **ORDER BY** is used for sorting, NULL is considered to be smaller than any other value. This action is the same as that in MySQL but different from that in Oracle.

Unlike **GROUP BY**, **ORDER BY** must be followed by the alias of the **SELECT** column. If the **SELECT** operation is performed on a column and the column alias is not specified, the column name is used as the column alias.

```

select * from sale_detail order by region;
-- Returns an error because order by is not used together with limit
select * from sale_detail order by region limit 100;
select region as r from sale_detail order by region limit 100;
-- Returns an error because ORDER BY is not followed by a column alias
select region as r from sale_detail order by r limit 100;

```

The number in [limit number] is a constant to limit the number of output rows. If you want to directly view the result of a `SELECT` statement without `LIMIT` from the screen output, you can view up to 10,000 rows. The upper limit of screen display varies by project. However, the upper limit can be controlled through the `setproject` console.

- `SORT BY` : For partial ordering, `DISTRIBUTE BY` must be added in front of the statement. `SORT BY` is used to partially sort the results of `DISTRIBUTE BY`. Aliases of `SELECT` output columns must be used.

```
select region from sale_detail distribute by region sort by region;
select region as r from sale_detail sort by region;
-- Returns an error and exits because no distribute by exists.
```

- `DISTRIBUTE BY` : Performs hash-based sharding on data by values of certain columns. Aliases of `SELECT` output columns must be used.

```
select region from sale_detail distribute by region;
-- Runs successfully because the column name is an alias
select region as r from sale_detail distribute by region;
-- Returns an error because DISTRIBUTE BY is not followed by a column alias
select region as r from sale_detail distribute by r;
```

`ORDER BY` and `GROUP BY` cannot be used together with `DISTRIBUTE BY` and `SORT BY`. Aliases of `SELECT` output columns must be used.

Note

- The keys of `ORDER BY`, `SORT BY`, and `DISTRIBUTE BY` must be output columns (namely, column aliases) of `SELECT` statements.
- In MaxCompute SQL parsing, `ORDER BY`, `SORT BY`, and `DISTRIBUTE BY` come after `SELECT` operations. Therefore, they can only accept the output columns of `SELECT` statements as keys.

3.8.2. SELECT Sequence

The actual logic execution sequence of `SELECT` statements written in compliance with the preceding `SELECT` syntax are different from the standard writing sequence.

See the following example:

```
SELECT key
      ,MAX(value)
FROM   src t
WHERE  value > 0
GROUP BY key
HAVING SUM(value) > 100
ORDER BY key
LIMIT 100
;
```

The actual logic execution sequence is FROM->WHERE->GROUP BY->HAVING->SELECT->ORDER BY->LIMIT .

- ORDER BY can only reference columns generated in the SELECT list rather than accessing columns in the FROM source table.
- The HAVING operation can access GROUP BY keys and aggregate functions. When the SELECT operation is performed, SELECT can only access group keys and aggregate functions rather than columns in the FROM source table if GROUP BY exists.
- The columns generated in the select list can only be referenced in by, rather than accessing the columns in the source table of from.

To avoid confusion, MaxCompute allows users to write a query statement by the execution sequence. For example, the preceding statement can be written as follows:

```
FROM   src t
WHERE  value > 0
GROUP BY key
HAVING SUM(value) > 100
SELECT key
      ,MAX(value)
ORDER BY key
LIMIT 100
;
```

example2:

```
SELECT shop_name
      ,total_price
      ,region
FROM   sale_detail
WHERE  total_price > 150
DISTRIBUTE BY region
SORT BY region
;
```

In fact, the order of logical execution is FROM->WHERE->SELECT->DISTRIBUTE BY->SORT BY .

3.8.3. UNION, INTERSECT, and EXCEPT

This topic describes the syntax of UNION ALL, UNION DISTINCT, INTERSECT ALL, INTERSECT DISTINCT, EXCEPT ALL, and EXCEPT DISTINCT, and provides examples of query statements.

The following are example syntax statements:

```
select_statement UNION ALL select_statement;
select_statement UNION [DISTINCT] select_statement;
select_statement INTERSECT ALL select_statement;
select_statement INTERSECT [DISTINCT] select_statement;
select_statement EXCEPT ALL select_statement;
select_statement EXCEPT [DISTINCT] select_statement;
select_statement MINUS ALL select_statement;
select_statement MINUS [DISTINCT] select_statement;
```

- **UNION:** Returns the union of two data sets.
- **INTERSECT:** Returns the intersection of two data sets.
- **EXCEPT:** Returns the supplementary set of a data set in another set.
- **MINUS:** Means the same as EXCEPT.

Examples

- **UNION ALL**

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4) t(a, b)
UNION ALL
SELECT * FROM VALUES (1, 2), (1, 4) t(a, b);
```

Result: The two data sets are merged.

a	b
1	2
1	4
1	2
1	2
3	4

- UNION DISTINCT

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4) t(a, b)
UNION
SELECT * FROM VALUES (1, 2), (1, 4) t(a, b);
```

Result: The result is the same as `SELECT DISTINCT * FROM (< Result of UNION ALL>) t;` .

a	b
1	2
1	4
3	4

- INTERSECT ALL

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 6) t(a, b)
INTERSECT ALL
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 7) t(a, b);
```

Result: Repeated rows are not deleted. Each repeated row can be regarded as a unique row with a unique number.

a	b
1	2
1	2
3	4

- INTERSECT DISTINCT

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 6) t(a, b)
INTERSECT
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (5, 7) t(a, b);
```

Result: The result is the same as `SELECT DISTINCT * FROM (< Result of INTERSECT ALL>) t;` .

```
+-----+-----+
| a     | b     |
+-----+-----+
| 1     | 2     |
| 3     | 4     |
+-----+-----+
```

- EXCEPT ALL

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8) t(a, b)
EXCEPT ALL
SELECT * FROM VALUES (3, 4), (5, 6), (5, 6), (9, 10) t(a, b);
```

Result: Repeated rows are not deleted. Each repeated row can be regarded as a unique row with a unique number.

```
+-----+-----+
| a     | b     |
+-----+-----+
| 1     | 2     |
| 1     | 2     |
| 3     | 4     |
| 7     | 8     |
+-----+-----+
```

- EXCEPT DISTINCT

```
SELECT * FROM VALUES (1, 2), (1, 2), (3, 4), (3, 4), (5, 6), (7, 8) t(a, b)
EXCEPT DISTINCT
SELECT * FROM VALUES (3, 4), (5, 6), (5, 6), (9, 10) t(a, b);
```

Result: The result is the same as `SELECT DISTINCT * FROM left_branch EXCEPT ALL SELECT DISTINCT * FROM right_branch;` .

```

+-----+-----+
| a      | b      |
+-----+-----+
| 1      | 2      |
| 7      | 8      |
+-----+-----+

```

Note

- The sequence of rows in the result set is not guaranteed to be in any specific order.
- The two data sets in one set operation must contain the same number of columns. If not, implicit type conversion is involved. Out of compatibility reasons, implicit type conversion of STRING and non-STRING data in set operations is disabled.
- Up to 256 tables can be contained in one set operation. If more than 256 tables are involved, an error is returned.
- If UNION is followed by CLUSTER BY, DISTRIBUTE BY, SORT BY, ORDER BY, or a LIMIT clause, when `set odps.sql.type.system.odps2=false;`, the clause applies to the last SELECT statement. When `set odps.sql.type.system.odps2=true;`, the clause applies to all the preceding UNION results. Consider the following example:

```

set odps.sql.type.system.odps2=true;
SELECT explode(array(3, 1)) AS (a) UNION ALL SELECT explode(array(0, 4, 2)) AS (a) ORD
ER
BY a LIMIT 3;

```

Result:

```

+-----+
| a      |
+-----+
| 0      |
| 1      |
| 2      |
+-----+

```

3.8.4. JOIN

You can use a JOIN operation to join two tables and obtain query results from the joined tables. In MaxCompute, JOIN operations include LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN, and INNER JOIN. MaxCompute supports multiple JOIN operations in an SQL statement, but does not support CROSS JOIN. A CROSS JOIN operation joins two tables without specifying conditions in the ON clause and returns the cartesian product of the input tables.

The command syntax is as follows:


```

join_table:
    table_reference join table_factor [join_condition]
    | table_reference {left outer|right outer|full outer|inner} join table_reference join_condition
table_reference:
    table_factor
    | join_table
table_factor:
    tbl_name [alias]
    | table_subquery alias
    | ( table_references )
join_condition:
    on equality_expression ( and equality_expression )

```

 Note

- `equality_expression` indicates an equality expression.
- If partition pruning conditions are specified in the `WHERE` clause, partition pruning takes effect for both tables. If partition pruning conditions are specified in the `ON` clause, partition pruning takes effect for the right table but not the left table. A full table scan is run for the left table. For more information, see [Reasonableness evaluation of partition pruning](#).

- **LEFT OUTER JOIN** : the LEFT JOIN operation. It returns all queried rows from the left table, including those that do not match any rows in the right table.

```

select a.shop_name as ashop, b.shop_name as bshop from shop a
    left outer join sale_detail b on a.shop_name=b.shop_name;
-- Both the shop and sale_detail tables have the shop_name column. Therefore, aliases are assigned to the shop_name columns in the SELECT clause to distinguish between the columns.

```

 **Note** If the values in some rows of the right table are duplicate, we recommend that you do not use `LEFT JOIN` consecutively many times. Otherwise, data bloat may occur during the `JOIN` operation and interrupt your jobs.

- **RIGHT OUTER JOIN** : the RIGHT JOIN operation. It returns all queried rows from the right table, including those that do not match any rows in the left table.

```

select a.shop_name as ashop, b.shop_name as bshop from shop a
    right outer join sale_detail b on a.shop_name=b.shop_name;

```

- **FULL OUTER JOIN** : the FULL JOIN operation. It returns all queried rows from both the left and right tables.

```
select a.shop_name as ashop, b.shop_name as bshop from shop a
full outer join sale_detail b on a.shop_name=b.shop_name;
```

In the following example, the left table is `test_table_a` and the right table is `test_table_b`. The `JOIN` clause is used to join tables. The `WHERE` clause is used to specify filter conditions. To query rows whose value of the `origin` column is equal to that of the `id` column and whose data timestamp is greater than 20180101, you can use `LEFT JOIN` to return all queried rows in the left table `test_table_a`. Normally, if you use `LEFT JOIN`, it returns all queried rows in the left table. If you use `RIGHT JOIN`, it returns all queried rows in the right table.

An error example is as follows:

```
SELECT s.id
      ,s.name
      ,s.origin
      ,d.value
FROM   test_table_a s
LEFT JOIN test_table_b d
ON     s.origin = d.id
WHERE  s.ds > "20180101" AND d.ds>"20180101";
```

If you use the `JOIN` clause before the `WHERE` clause, the `JOIN` operation is performed first. Then, the results obtained from the `JOIN` operation are filtered based on the conditions specified by the `WHERE` clause. The final results are the intersection of two tables but not all queried rows in the left table.

To return all queried rows in the left table, you can modify the SQL statement as follows:

```
SELECT s.id
      ,s.name
      ,s.origin
      ,d.value
FROM   (select * from test_table_a where ds > "20180101" ) s
LEFT JOIN (select * from test_table_b where ds > "20180101") d
ON     s.origin = d.id;
```

INNER JOIN : returns a result table that combines the queried rows from two tables that meet the conditions specified by the `ON` clause. The keyword `INNER` can be omitted.

```
select a.shop_name from shop a inner join sale_detail b on a.shop_name=b.shop_name;
select a.shop_name from shop a join sale_detail b on a.shop_name=b.shop_name;
```

If you specify multiple conditions in the ON clause, you must specify equivalent conditions and use `and` to connect these conditions. You can specify a non-equivalent condition or use `or` to connect multiple conditions in the ON clause only for **MAPJOIN**.

```
select a.* from shop a full outer join sale_detail b on a.shop_name=b.shop_name
    full outer join sale_detail c on a.shop_name=c.shop_name;
-- MaxCompute supports multiple JOIN operations in an SQL statement.
select a.* from shop a join sale_detail b on a.shop_name != b.shop_name;
-- MaxCompute does not support a non-equivalent condition and returns an error.
```

MaxCompute supports the following implicit `JOIN` operation:

```
SELECT * FROM table1, table2 WHERE table1.id = table2.id;
-- The effect of this statement is equivalent to the following statement:
SELECT * FROM table1 JOIN table2 ON table1.id = table2.id;
```

MaxCompute also supports the **NATURAL JOIN** operation, in which the conditions used to join two tables are automatically determined based on the common fields in the two tables.

MaxCompute supports **OUTER NATURAL JOIN**. You can use the **USING** clause so that the JOIN operation returns common fields only once. For example, to join the `src` table that contains the `key1`, `key2`, `a1`, and `a2` fields and the `src2` table that contains the `key1`, `key2`, `b1`, and `b2` fields, you can run the following statement:

```
SELECT * FROM src NATURAL JOIN src2;
-- Both the src and src2 tables contain the key1 and key2 fields. In this case, the preceding statement i
s equivalent to the following statement:
SELECT src.key1 as key1, src.key2 as key2, src.a1, src.a2, src2.b1, src2.b2 FROM src INNER JOIN src2 ON s
rc.key1 = src2.key1 AND src.key2 = src2.key2;
```

You can use parentheses `()` to specify the priority of JOIN operations. The JOIN operation enclosed in parentheses `()` has a higher priority. In the following example, the `src2 JOIN src3` operation is performed first and the `src` JOIN operation is then performed on the obtained results.

```
SELECT * FROM src JOIN (src2 JOIN src3 on xxx) ON yyy;
```

3.8.5. SEMI JOIN

MaxCompute supports two types of SEMI JOIN operations: **LEFT SEMI JOIN** and **LEFT ANTI JOIN**. In SEMI JOIN, the right table does not appear in the result set and is only used to filter data in the left table.

LEFT SEMI JOIN

A **LEFT SEMI JOIN** returns all rows from the left table that match a row in the right table. For example, if the **id** value of a row in **mytable1** appears in the **id** column in **mytable2**, this row is returned in the result set.

To implement the preceding logic, execute the following statement:

```
SELECT * from mytable1 a LEFT SEMI JOIN mytable2 b on a.id=b.id;
```

In this example, only rows in **mytable1** with IDs that appear in **mytable2** are returned in the result set.

LEFT ANTI JOIN

A **LEFT ANTI JOIN** returns all rows from the left table that do not have a match in the right table. For example, if the **id** value of a row in **mytable1** does not appear in the **id** column in **mytable2**, this row is returned in the result set.

To implement the preceding logic, execute the following statement:

```
SELECT * from mytable1 a LEFT ANTI JOIN mytable2 b on a.id=b.id;
```

In this example, only rows in **mytable1** with IDs that do not appear in **mytable2** are returned in the result set.

3.8.6. MAPJOIN HINT

This topic describes how to use a **MAPJOIN** statement to join a large table with one or more small tables. A **MAPJOIN** operation is faster than common **JOIN** operations.

When the volume of data is small, **MAPJOIN** accelerates the execution process by using SQL to load all the specified small tables into the program memory through the **JOIN** operation.

 **Note** Note the following points when you use a MAPJOIN statement:

- The left table of a `LEFT OUTER JOIN` clause must be a large table.
- The right table of a `RIGHT OUTER JOIN` clause must be a large table.
- Both the left and right tables of an `INNER JOIN` clause can be large tables.
- `MAPJOIN` cannot be used in a `FULL OUTER JOIN` clause.
- `MAPJOIN` supports small tables in subqueries.
- If you need to reference a small table or a subquery when you perform `MAPJOIN`, you must reference the alias of the table or subquery.
- In `MAPJOIN`, you can use non-equi joins or combine multiple conditions by using `OR`. You can calculate the Cartesian product by leaving the `ON` condition unspecified or by using `mapjoin on 1 = 1`, for example, `select /* + mapjoin(a) */ a.id from shop a join table_name b on 1=1`. However, this calculation method may lead to data expansion.
- MaxCompute allows you to specify a maximum of 128 small tables for the `MAPJOIN` operation. If the number of small tables exceeds 128, a syntax error is returned. In `MAPJOIN HINT`, multiple small tables are separated with commas (,), for example, `/* + mapjoin(a,b,c) */`.
- In `MAPJOIN`, the total memory occupied by small tables cannot exceed 640 MB. MaxCompute compresses your data before storage. As a result, the data volume of small tables is sharply increased after they are loaded into the memory. 640 MB indicates the maximum data volume after small tables are loaded into the memory.

Example

```
select /* + mapjoin(a) */
  a.shop_name,
  b.customer_id,
  b.total_price
from shop a join sale_detail b
on a.shop_name = b.shop_name;
```

In MaxCompute SQL, you cannot use non-equi joins or the `OR` logic in the `ON` condition for a common `JOIN` operation. However, you can do this in `MAPJOIN`. An example is as follows:

```
select /*+ mapjoin(a) */
  a.total_price,
  b.total_price
from shop a join sale_detail b
on a.total_price < b.total_price or a.total_price + b.total_price < 500;
```

3.8.7. HAVING clause

HAVING clauses are used because the Where keyword of MaxCompute SQL cannot be used together with aggregate functions.

Function definition.

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value
```

Example.

A table named Orders contains four fields: Customer, OrderPrice, Order_date, and Order_id. To query customers whose OrderPrice is smaller than 2,000, The SQL statement is as follows:

```
SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer
HAVING SUM(OrderPrice)<2000
```

3.8.8. Grouping Sets

This topic describes how to use the GROUPING SETS clause when you need to execute multiple UNION ALL clauses for scenarios where you need to aggregate and analyze data of multiple dimensions.

GROUPING SETS by MaxCompute is an extension to the GROUP BY clause in the SELECT statement. You can group results in various ways by using GROUPING SETS without executing multiple SELECT statements. This can allow the MaxCompute engine to produce better implementation plans with higher performance.

 **Note** A number of examples in this topic are demonstrated by using MaxCompute Studio. We recommend that you install MaxCompute Studio by following the instructions provided in [Install IntelliJ IDEA](#) before you proceed with subsequent operations.

Example

1. Prepare data.

```
create table requests LIFECYCLE 20 as
select * from values
  (1, 'windows', 'PC', 'Beijing'),
  (2, 'windows', 'PC', 'Shijiazhuang'),
  (3, 'linux', 'Phone', 'Beijing'),
  (4, 'windows', 'PC', 'Beijing'),
  (5, 'ios', 'Phone', 'Shijiazhuang'),
  (6, 'linux', 'PC', 'Beijing'),
  (7, 'windows', 'Phone', 'Shijiazhuang')
as t(id, os, device, city);
```

2. Use one of the following two methods to group data:

- Execute multiple SELECT statements.

```
SELECT NULL, NULL, NULL, COUNT(*)
FROM requests
UNION ALL
SELECT os, device, NULL, COUNT(*)
FROM requests GROUP BY os, device
UNION ALL
SELECT null, null, city, COUNT(*)
FROM requests GROUP BY city;
```

- Use GROUPING SETS.

```
SELECT os,device, city ,COUNT(*)
FROM requests
GROUP BY os, device, city GROUPING SETS((os, device), (city), ());
```

Data is aggregated as follows:

```
+---+-----+-----+-----+
| os | device | city | cnt   |
+---+-----+-----+-----+
| NULL | NULL  | NULL | 7     |
| NULL | NULL  | Beijing | 4    |
| NULL | NULL  | Shijiazhuang | 3    |
| ios | Phone | NULL | 1     |
| linux | PC   | NULL | 1     |
| linux | Phone | NULL | 1     |
| windows | PC  | NULL | 3     |
| windows | Phone | NULL | 1     |
+---+-----+-----+-----+
```

 **Note** Expressions not used in GROUPING SETS use NULL as placeholders. You can execute UNION statements on grouping sets, such as the city column in lines 1 through 5.

CUBE and ROLLUP

CUBE and ROLLUP are special GROUPING SETS functions.

CUBE lists all possible combinations of specified columns as grouping sets. ROLLUP aggregates data by level to produce grouping sets.

Example code:

```
GROUP BY CUBE(a, b, c) is equal to GROUPING SETS((a,b,c),(a,b),(a,c),(b,c),(a),(b),(c),()).
```

```
GROUP BY ROLLUP(a, b, c) is equal to GROUPING SETS((a,b,c),(a,b),(a)).
```

GROUPING and GROUPING_ID

NULL is used as placeholders in grouping sets, but it can also be a value that is manually entered. In the code, however, placeholder NULLs are indistinguishable from value NULLs. The GROUPING function is provided to address this issue.

GROUPING allows you to specify the name of a column as a parameter. If the specified lines are aggregated based on a column whose name is used as a parameter in this function, 0 is returned, indicating that NULL is an entered value. Otherwise, 1 is returned, indicating that NULL is a placeholder.

GROUPING_ID can be used to specify the names of one or more columns as parameters. The GROUPING results in these columns are formed into integers by using BitMap.

Example code:

```
SELECT a,b,c ,COUNT(*),  
GROUPING(a) ga, GROUPING(b) gb, GROUPING(c) gc, GROUPING_ID(a,b,c) groupingid  
FROM VALUES (1,2,3) as t(a,b,c)  
GROUP BY CUBE(a,b,c);
```

Data is aggregated as follows:

a	b	c	_c3	ga	gb	gc	groupingid
NULL	NULL	NULL	1	1	1	1	7
NULL	NULL	3	1	1	1	0	6
NULL	2	NULL	1	1	0	1	5
NULL	2	3	1	1	0	0	4
1	NULL	NULL	1	0	1	1	3
1	NULL	3	1	0	1	0	2
1	2	NULL	1	0	0	1	1
1	2	3	1	0	0	0	0

3.8.9. SELECT TRANSFORM

This topic describes how to use the `SELECT TRANSFORM` statement to perform operations that MaxCompute SQL does not directly support. Specifically, by using `SELECT TRANSFORM`, you can start a specified child process and enter data of a required format into the child process through standard input (stdin). Then, you can parse the standard output (stdout) of the child process to obtain the final output. This process does not require you to compile user-defined functions (UDFs).

Example code:

```
SELECT TRANSFORM(arg1, arg2 ...)
(Row FORMAT DELIMITED (FIELDS TERMINATED BY field_delimiter (ESCAPED BY character_escape)?)?
(LINES SEPARATED BY line_separator)?
(NULL DEFINED AS null_value)?)?
USING 'unix_command_line'
(RESOURCES 'res_name' (',' 'res_name')*)?
( AS col1, col2 ...)?
(Row FORMAT DELIMITED (FIELDS TERMINATED BY field_delimiter (ESCAPED BY character_escape)?)?
(LINES SEPARATED BY line_separator)? (NULL DEFINED AS null_value)?)?
```

Parameters:

- The `SELECT TRANSFORM` keyword can be replaced with the `MAP` or `REDUCE` keyword, but the meaning remains unchanged. However, to make the syntax clearer, we recommend that you use `SELECT TRANSFORM`.
- `arg1,arg2...` indicates arguments in the `TRANSFORM` clause. The format of arguments in the `TRANSFORM` clause is similar to that of items in the `SELECT` clause. In the default format, the results of expressions for each argument are combined by using `\t` after they are implicitly converted into strings, and then they are entered into the specified child process. The default

format is configurable. For more information, see the **ROW FORMAT** clause described in the following section.

- **USING** : specifies the command for starting a child process.

 Note

- In most MaxCompute SQL statements, the **USING** clause can only specify resources. However, with the **SELECT TRANSFORM** statement, the **USING** clause can specify commands to ensure compatibility with the syntax of Hive.
- The format of the **USING** clause is similar to the syntax of a Shell script. However, a Shell script is actually not executed to start the child process, and the child process is created according to the command input. Therefore, a number of Shell functions such as input and output redirection, pipe, and loop are unavailable. A Shell script can be used as the command for a child process if needed.

- **RESOURCES** : specifies the resources that the specified child process can access. You can use one of the following two methods to specify resources:
 - Use the **RESOURCES** clause, for example, `using 'sh foo.sh bar.txt' Resources 'foo.sh','bar.txt'`.
 - Add the `set odps.sql.session.resources=foo.sh,bar.txt;` clause before SQL statements. This clause takes effect globally once it is specified, which means that all **SELECT TRANSFORM** statements can access the resources specified by this clause.
- **ROW FORMAT** : specifies the input or output format.

The syntax includes two **ROW FORMAT** clauses: One specifies the input format, and the other specifies the output format. By default, `\t` is used to separate columns, `\n` is used to separate rows, and `Null` is represented by `\N`.

 Note

- Only one character is accepted by `field_delimiter`, `character_escape`, and `line_separator`. If you specify a string, the first character in the string takes priority over the others.
- Hive specifies the syntax for each format. Such syntaxes as `inputRecordReader`, `outputRecordReader`, and `Serde` are supported by MaxCompute. To use these formats, you need to enable compatibility with Hive by adding `set odps.sql.hive.compatible=true;` preceding SQL statements. For more information about the syntaxes supported by Hive, see [Hive](#).
- If you specify a syntax such as `inputRecordReader` or `outputRecordReader` supported by Hive, statements may be executed at lower speeds.

- **AS** : specifies output columns.
 - You can specify data types in the **AS** clause, for example, `as(col1:bigint, col2:boolean)`. If you do not specify data types, for example, `as(col1, col2)`, strings are returned by default.

- The output is obtained by parsing the stdout of the child process. If the specified data types do not include `STRING`, the system implicitly calls the `CAST` function, which may encounter runtime exceptions.
- You are not allowed to specify data types for only some of the specified columns, for example, `as(col1, col2:bigint)`.
- If you skip the `AS` clause, the field preceding the first `\t` in the stdout is a key, and all the following parts are values, equivalent to `as(key, value)`.

Call Shell scripts

In the case that you execute a Shell script to generate 50 lines of data starting from 1 to 50, the output of the data field is as follows:

```
SELECT TRANSFORM(script) USING 'sh' AS (data)
FROM (
    SELECT 'for i in `seq 1 50`; do echo $i; done' AS script
) t
;
```

The Shell commands are used as the input of the `TRANSFORM` clause.

`SELECT TRANSFORM` is not only a language extension. Simple functions such as `AWK`, `Python`, `Perl`, and `Shell` allow you to compile scripts in commands, so that you do not need to compile independent script files or upload resources.

Call Python scripts

`Python` is a simple function that can allow you to compile scripts in commands so that you do not need to compile independent script files or upload resources. The following is an example of how to call `Python` scripts.

1. Compile a `Python` script file. In this example, the file name is *myplus.py*

```
#!/usr/bin/env python
import sys
line = sys.stdin.readline()
while line:
    token = line.split('\t')
    if (token[0] == '\\N') or (token[1] == '\\N'):
        print '\\N'
    else:
        print int(token[0]) + int(token[1])
    line = sys.stdin.readline()
```

2. Add the `Python` script file as a resource to MaxCompute.

```
add py ./myplus.py -f;
```

 **Note** You can add resources by using the DataWorks console.

3. Use SELECT TRANSFORM to call the resource.

Create table testdata(c1 bigint,c2 bigint); - Creates a test table.

insert into Table testdata values (1,4),(2,5),(3,6); - Inserts test data into the test table.

- Execute the SELECT TRANSFORM statement:

```
SELECT
TRANSFORM (testdata.c1, testdata.c2)
USING 'python myplus.py' resources 'myplus.py'
AS (result bigint)
FROM testdata;
```

- Or:

```
set odps.sql.session.resources=myplus.py;
SELECT TRANSFORM (testdata.c1, testdata.c2)
USING 'python myplus.py'
AS (result bigint)
FROM testdata;
```

The returned information is as follows:

```
+-----+
| cnt |
+-----+
| 5 |
| 7 |
| 9 |
+-----+
```

Python scripts do not require MaxCompute to run in a Python framework, and they are not subject to any format requirements.

In MaxCompute, Python commands can be used as the input of the **TRANSFORM** clause. For example, you can call Shell scripts by running Python commands.

```
SELECT TRANSFORM('for i in xrange(1, 50): print i;') USING 'python' AS (data);
```

Call Java scripts

Calling a Java script is similar to calling a Python script. In this example, you need to compile a Java script file, export it as a .jar package, and then run the add command to add the .jar package as a resource to MaxCompute. The resource will be called by using SELECT TRANSFORM.

1. Compile a Java script file, and export it as a .jar package. In this example, the name of the .jar package is *Sum.jar*.

```
package com.aliyun.odps.test;
import java.util.Scanner
public class Sum {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            String s = sc.nextLine();
            String[] tokens = s.split("\t");
            if (tokens.length < 2) {
                throw new RuntimeException("illegal input");
            }
            if (tokens[0].equals("\N") || tokens[1].equals("\N")) {
                System.out.println("\N");
            }
            System.out.println(Long.parseLong(tokens[0]) + Long.parseLong(tokens[1]));
        }
    }
}
```

2. Add the .jar package as a resource to MaxCompute.

```
add jar ./Sum.jar -f;
```

3. Use SELECT TRANSFORM to call the resource.

```
Create table testdata(c1 bigint,c2 bigint); - Creates a test table.
insert into Table testdata values (1,4),(2,5),(3,6); - Inserts test data into the test table.
- Execute the SELECT TRANSFORM statement:
SELECT TRANSFORM(testdata.c1, testdata.c2) USING 'java -cp Sum.jar com.aliyun.odps.test.Sum' re
sources 'Sum.jar' from testdata;
- Or:
set odps.sql.session.resources=Sum.jar;
SELECT TRANSFORM(testdata.c1, testdata.c2) USING 'java -cp Sum.jar com.aliyun.odps.test.Sum' FR
OM testdata;
```

The returned information is as follows:

```
+-----+
| cnt |
+-----+
| 5 |
| 7 |
| 9 |
+-----+
```

You can run most Java utilities by using the preceding method.

 **Note** Although user-defined table-valued function (UDTF) frameworks are provided for Java and Python, compiling code by using `SELECT TRANSFORM` is easier. `SELECT TRANSFORM` is a simpler process because it is not subject to any format requirements and can be called offline. The paths for Java and Python offline scripts can be obtained from the `JAVA_HOME` and `PYTHON_HOME` environment variables, respectively.

Call scripts of other languages

In addition to the language extensions mentioned above, `SELECT TRANSFORM` supports commonly used Unix command and script interpreters such as AWK and Perl.

An example of how to output column 2 by calling AWK:

```
SELECT TRANSFORM(*) USING "awk '{print $2}'" as (data) from testdata;
```

An example of calling Perl:

```
SELECT TRANSFORM (testdata.c1, testdata.c2) USING "perl -e 'while($input = <STDIN>){print $input;}'" FROM testdata;
```

 **Note** Currently PHP and Ruby are not deployed in the MaxCompute cluster. Therefore, MaxCompute does not allow you to call PHP or Ruby scripts.

Call scripts in tandem

`SELECT TRANSFORM` allows you to call scripts in tandem. For example, you can use `distribute by` and `sort by` to preprocess data.

```
SELECT TRANSFORM(key, value) USING 'cmd2' from
(
  SELECT TRANSFORM(*) USING 'cmd1' from
  (
    SELECT * FROM data distribute by col2 sort by col1
  ) t distribute by key sort by value
) t2;
```

You can use either the map or reduce keywords with the same results.

```
@a := select * from data distribute by col2 sort by col1;
@b := map * using 'cmd1' distribute by col1 sort by col2 from @a;
reduce * using 'cmd2' from @b;
```

SELECT TRANSFORM performance

The performance of SELECT TRANSFORM and UDTF varies depending on the specific scenario. In general, the performance of SELECT TRANSFORM is optimal when the data size is comparatively small, and the performance of UDTF is optimal when the data size is large.

SELECT TRANSFORM is easier to use, and therefore is more suitable for adhoc data analysis.

Benefits of UDTF

- In UDTF, the output and input must follow the specified data types, and therefore UDTF does not require data type conversion as is required by SELECT TRANSFORM.
- In UDTF, processes are not suspended if the operating system pipe is empty or fully occupied as is the case for SELECT TRANSFORM.
- In UDTF, constant parameters do not need to be transmitted, which is required by SELECT TRANSFORM.

Benefits of SELECT TRANSFORM

- SELECT TRANSFORM supports a child process and a parent process and therefore can take advantage of multiple cores in servers.
- SELECT TRANSFORM calls underlying systems to read and write the data to be transmitted, which allows it to have performance that is higher than that of Java.
- SELECT TRANSFORM supports tools such as AWK and can run native code. This offers SELECT TRANSFORM more advantages compared to Java.

3.9. Script Mode SQL

This topic describes Script Mode SQL and how to run this language.

 **Note** The fees information detailed in the instructions section of [Other operations](#) for Script Mode SQL is provided as reference purposes only. The fees displayed on the billing console are the actual fees charged to your account. For more information, see [View billing details](#).

What is Script Mode SQL

MaxCompute supports Script Mode SQL. In script mode type of SQL, statements are compiled as a whole according to the logic in an SQL script file. Then the script file is submitted to MaxCompute to generate one execution plan, so that these statements are scheduled in one queue and run all at once.

Script Mode SQL is used similarly to other common programming languages. This language can assist in compiling scripts more efficiently by eliminating the need to organize statements for these scripts.

Script Mode SQL supports the following tools:

- MaxCompute Studio: For more information, see [What is Studio](#).
- MaxCompute command line interface (CLI): For more information, see [Install and configure the odpscmd client](#).

Syntax


```

--set
set odps.sql.type.system.odps2=true;
[set odps.stage.reducer.num=xxx;]
[...]
--ddl
create table table1 xxx;
[create table table2 xxx;]
[...]
--dml
@var1 := SELECT [ALL | DISTINCT] select_expr, select_expr, ...
      FROM table3
      [WHERE where_condition];
@var2 := SELECT [ALL | DISTINCT] select_expr, select_expr, ...
      FROM table4
      [WHERE where_condition];
@var3 := SELECT [ALL | DISTINCT] var1.select_expr, var2.select_expr, ...
      FROM @var1 join @var2 on ...;
INSERT OVERWRITE|INTO TABLE [PARTITION (partcol1=val1, partcol2=val2 ...)]
      SELECT [ALL | DISTINCT] select_expr, select_expr, ...
      FROM @var3;
[@var4 := SELECT [ALL | DISTINCT] var1.select_expr, var.select_expr, ... FROM @var1
      UNION ALL | UNION
      SELECT [ALL | DISTINCT] var1.select_expr, var.select_expr, ... FROM @var2;
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table_name
      AS
      SELECT [ALL | DISTINCT] select_expr, select_expr, ...
      FROM var4;]
[...]

```

- Script Mode SQL supports **SET** statements, data manipulation language (DML) statements, and data definition language (DDL) statements (excluding such statements as **DESC** and **SHOW** whose results are displayed on your screen).
- A complete script must start with **SET** statements, followed by DDL statements, and end with DML statements. One part may either be empty or contain several statements. Different types of statements cannot co-occur in one part.
- The at sign **@** indicates a variable.
- A script can contain up to one statement, such as a **SELECT** statement, whose result is displayed on your screen. If one script contains two or more such statements, errors are more likely to occur. Therefore, we recommend that you do not run such **SELECT** statements.

- A script can contain only one `CREATE TABLE AS` statement, and must end with this statement. We recommend that you separate the `CREATE TABLE AS` statement from `INSERT` statements.
- If a statement fails to run, all the other statements in the same script also fail.
- A job is generated to process data only after all the input data is prepared.
- If a table is written and then read, the following error occurs:

```
insert overwrite table src2 select * from src where key > 0;
@a := select * from src2;
select * from @a;
```

To avoid this error, modify your SQL script as follows:

```
@a := select * from src where key > 0;
insert overwrite table src2 select * from @a;
select * from @a;
```

Example

```
create table if not exists dest(key string , value bigint) partitioned by (d string);
create table if not exists dest2(key string,value bigint) partitioned by (d string);
@a := select * from src where value >0;
@b := select * from src2 where key is not null;
@c := select * from src3 where value is not null;
@d := select a.key,b.value from @a left outer join @b on a.key=b.key and b.value>0;
@e := select a.key,c.value from @a inner join @c on a.key=c.key;
@f := select * from @d union select * from @e union select * from @a;
insert overwrite table dest partition (d='20171111') select * from @f;
@g := select e.key,c.value from @e join @c on e.key=c.key;
insert overwrite table dest2 partition (d='20171111') SELECT * from @g;
```

Scenarios

We recommend that you use Script Mode SQL in the two following scenarios:

- You require multiple layers of query clauses to be nested into one statement, or require a complicated script to be split into multiple simpler statements.
- You require the input data obtained from multiple sources at long intervals to be combined into a script. However, the input data cannot be combined by using table variables alone.

Run Script Mode SQL by using MaxCompute Studio

Before running Script Mode SQL, you must first install MaxCompute Studio, add your project link, and create a MaxCompute SQL script file. For more information, see [Install IntelliJ IDEA](#), [Project space connection management](#), and [Create MaxCompute Script module](#).

After submitting your script, you can view the corresponding plan for running the script in a directed acyclic graph (DAG). This is the case even though the script contains multiple statements.

Run Script Mode SQL by using MaxCompute Console (odpscmd)

For the following examples, we recommend that you use MaxCompute CLI (odpscmd) v0.27.0 or a later version, which can be downloaded from [Directory Listing For odpscmd](#). After installing the odpscmd tool, you can use the `-s` parameter to submit your script.

To edit the *myscript.sql* source code file in script mode, run the following command:

```
odpscmd -s myscript.mxql;
```

 **Note** `-s` is a parameter similar to `-f` and `-e` rather than a command in the odpscmd tool. The odpscmd tool does not support Script Mode SQL or table variables.

Run Script Mode SQL by using DataWorks

You can create a node that runs in script mode, as shown in the following figure.

On this node, you can compile your script in script mode and then click Run on the toolbar to submit your script to MaxCompute. You can visit the URL of Logview in the command output to obtain the script execution plan and result.

3.10. Common table expression (CTE)

MaxCompute supports CTEs in standard SQL to improve the readability and execution efficiency of SQL statements.

Syntax structure of CTE.

```
WITH
  cte_name AS

  cte_query

[,cte_name2 AS

  cte_query2

,.....]
```

- `cte_name` refers to the CTE name, which must be unique in current WITH clause. The `cte_name` identifier in any position of the query indicates the CTE.
- `cte_query` is a SELECT statement, whose result set is used to populate the CTE.

Example:

```

INSERT OVERWRITE TABLE srcp PARTITION (p='abc')
SELECT * FROM (
    SELECT a.key, b.value
    FROM (
        SELECT * FROM src WHERE key IS NOT NULL ) a
    JOIN (
        SELECT * FROM src2 WHERE value > 0 ) b
    ON a.key = b.key
) c
UNION ALL
SELECT * FROM (
    SELECT a.key, b.value
    FROM (
        SELECT * FROM src WHERE key IS NOT NULL ) a
    LEFT OUTER JOIN (
        SELECT * FROM src3 WHERE value > 0 ) b
    ON a.key = b.key AND b.key IS NOT NULL
)d;

```

A JOIN clause is written on both sides of UNION at the top layer, and same queries are formed on the left table of JOIN. You must repeat this code if writing subqueries.

The preceding statement can be rewritten as follows using the CTE:

```

with
a as (select * from src where key is not null),
b as (select * from src2 where value>0),
c as (select * from src3 where value>0),
d as (select a.key,b.value from a join b on a.key=b.key ),
e as (select a.key,c.value from a left outer join c on a.key=c.key and c.key is not null )
insert overwrite table srcp partition (p='abc')
select * from d union all select * from e;

```

After rewriting, the subquery corresponding to "a" only need to be rewritten once, and then can be reused subsequently. The WITH clause in the CTE specifies multiple subqueries that can be repeatedly used like variables in the entire statement. Besides being reused, subqueries do not have to be repeatedly nested.

3.11. Explain

The Explain operation of MaxCompute SQL helps to display the description of the final execution plan structure corresponding to a DML statement. The execution plan is the program used at the final stage to run SQL semantics.

Function definition.

```
EXPLAIN <DML query>;
```

The execution result of 'explain' includes the following:

- The dependency structure of all the tasks corresponding to this DML statement.
- All task dependency structures in a task.
- All operator dependency structures in a task.

For examples.

```
EXPLAIN
SELECT abs(a.key), b.value FROM src a JOIN src1 b ON a.value = b.value;
```

The output result of Explain consists of the following parts:

- The dependency between jobs: job0 is root job , As the query requires one job (job0), only one row of information is required.
- The dependency between tasks:

```
In Job job0:
root Tasks: M1_Stg1, M2_Stg1
J3_1_2_Stg1 depends on: M1_Stg1, M2_Stg1
```

Job0 contains three tasks, among which M1_Stg1 and M2_Stg1 are run first, followed by J3_1_2_Stg1.

The naming rules of tasks are as follows:

- MaxCompute contains four types of tasks: MapTask, ReduceTask, JoinTask, and LocalWork.
- The first letter of a task name represents the current task type. For example, M2Stg1 is a MapTask.
- The number following the first letter represents the current task ID, which must be unique in all tasks corresponding to the current query.
- The numbers separated by underscores (_) represent the direct dependencies of the current task. For example, J3_1_2_Stg1 indicates that the current task (whose ID is 3) depends on two tasks whose IDs are 1 and 2.
- The third part is the operator structure in the task. The operator string describes the execution semantics of a task:

In Task M1_Stg1:

Data source: yudi_2.src # Data source describes the input content of the current task

TS: alias: a # TableScanOperator

RS: order: + # ReduceSinkOperator

keys:

a.value

values:

a.key

partitions:

a.value

In Task J3_1_2_Stg1:

JOIN: a INNER JOIN b # JoinOperator

SEL: Abs(UDFToDouble(a._col0)), b._col5 # SelectOperator

FS: output: None # FileSinkOperator

In Task M2_Stg1:

Data source: yudi_2.src1

TS: alias: b

RS: order: +

keys:

b.value

values:

b.value

partitions:

b.value

- Description of operators:
 - **TableScanOperator:** Describes the logic of FROM statement blocks in a Query statement. The input table name (alias) is displayed in the EXPLAIN results.
 - **SelectOperator:** Describes the logic of SELECT statement blocks in a QUERY statement. The columns to be passed to the next operator are displayed in the Explain results, separated by commas (,).
 - If column references are to be passed, `< alias >.< column_name >` is displayed
 - If expression results are to be transmitted, they are displayed as functions, for example, `func1(arg1_1, arg1_2, func2(arg2_1, arg2_2))`.
 - If constants are to be passed, the values are directly displayed.
 - **FilterOperator:** Describes the logic of WHERE statement blocks in a QUERY statement. A WHERE condition expression is displayed in the Explain results, with the display rules similar to those of SelectOperator.
 - **JoinOperator:** Describes the logic of JOIN statement blocks in a QUERY statement. Both the tables to be joined and the JOIN method are displayed in the Explain results.
 - **GroupByOperator:** Describes the logic of aggregate operations. This structure is displayed if an aggregate function is used in a QUERY statement. The aggregate function content is displayed in the Explain results.
 - **ReduceSinkOperator:** Describes the logic of data distribution operations between tasks. If the result of the current task is to be passed to another task, ReduceSinkOperator must be used at the end of the current task to perform the data distribution operation. The sorting method of output results, distributed keys, values, and columns used to calculate the hash value are displayed in the Explain results.
 - **FileSinkOperator:** Describes the storage operation of final data. If Insert statement blocks exist in the QUERY statement, the target table name is displayed in the Explain results.
 - **LimitOperator:** Describes the logic of Limit statement blocks in a QUERY statement. The number of LIMIT is displayed in the Explain results.
 - **MapJoinOperator:** Similar to JoinOperator, it describes JOIN operations in large tables.

 **Note** If a QUERY statement is so complicated that Explain has too many results, API restrictions are triggered, which leads to incomplete display of Explain results. In this case, you can split the QUERY and perform the Explain operation on each part to understand the job structure.

3.12. LOAD

This topic describes how to use LOAD statements to import data into a table or a partition of a table from external storage, such as Object Storage Service (OSS).

LOAD data import

Precautions

MaxCompute must be authorized to access data from external storage. The methods used to authorize MaxCompute to access external data by executing LOAD statements are the same as the methods used to authorize MaxCompute external tables. You can use one of the following methods for authorization:

- Specify the AccessKey ID and AccessKey secret in the data storage path. Use OSS as an example:

```
'oss://<yourAccessKeyId>:<yourAccessKeySecret>@oss-cn-hangzhou-zmf.aliyuncs.com/my_bucket_id/my_location/'
```

 **Note** If you use this method, the AccessKey ID and AccessKey secret are displayed in plaintext. This method entails security risks and is not recommended.

- Use STS. For more information, see [STS authorization](#).

Syntax

- Use an extractor or a storage handler to import data.

```
LOAD OVERWRITE|INTO TABLE table_name [PARTITION(partcol1=val1, partcol2=val2 ...)]
FROM LOCATION external_location
[STORED BY StorageHandler]
[WITH SERDEPROPERTIES (Options)];
```

- Import data in open-source formats.

```
LOAD OVERWRITE|INTO TABLE table_name [PARTITION(partcol1=val1, partcol2=val2 ...)]
FROM LOCATION external_location
[ROW FORMAT SERDE '<serde class>'
[WITH SERDEPROPERTIES ('odps.properties.rolearn'='${roleran}' [, 'name2'='value2',...])]
]
STORED AS <file format>;
```

Description

LOAD statements can append data to or overwrite data in a table or partition.

Parameters

- table_name**: the name of the destination table to which you want to append data. You must create a destination table before you append data to it. The schema of the destination table must be the same as the format of the external data.
- LOAD INTO**: appends data to a table or partition.
- LOAD OVERWRITE**: clears the existing data in a table and then inserts data into the table or its partition.
- SORTED BY**: specifies the name of the storage handler. You can use this clause in the same way as you use it for MaxCompute external tables. For more information, see [Access OSS data by using the built-in extractor](#).
- STORED AS**: specifies the format of the imported data file, such as ORC, PARQUET, RCFILE, SEQUENCEFILE, and TEXTFILE. You can use this clause in the same way as you use it for MaxCompute external tables. For more information, see [Process OSS data stored in open-source formats](#).

- **ROW FORMAT SERDE:** This clause is required only when you use special file formats, such as TEXTFILE. You can use this clause in the same way as you use it for MaxCompute external tables. For more information, see [Process OSS data stored in open-source formats](#).

The mappings between file formats and SerDe classes are as follows:

- SEQUENCEFILE: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
 - TEXTFILE: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
 - RCFILE: org.apache.hadoop.hive.serde2.columnar.LazyBinaryColumnarSerDe
 - ORC: org.apache.hadoop.hive ql.io.orc.OrcSerde
 - ORCFILE: org.apache.hadoop.hive ql.io.orc.OrcSerde
 - PARQUET: org.apache.hadoop.hive ql.io.parquet.serde.ParquetHiveSerDe
 - AVRO: org.apache.hadoop.hive.serde2.avro.AvroSerDe
- **SERDEPROPERTIES:** If you use STS to authorize MaxCompute, you must use this parameter to specify the value of `odps.properties.rolearn`, which is the Alibaba Cloud Resource Name (ARN) specified in the AliyunODPSDefaultRole RAM role. For more information, see [STS authorization](#).

Examples

- Import text files to a destination table.

```
LOAD overwrite table tsv_load_tbl
from
LOCATION 'oss://accessId:accessKey@oss-cn-hangzhou-zmf.aliyuncs.com/my_bucket_id/my_location/'
STORED BY 'com.aliyun.odps.TsvStorageHandler'
WITH SERDEPROPERTIES ('delimiter'='\t');
```

- Import data in open-source formats to a destination table.

```
-- Create destination table oss_load_static_part.
CREATE TABLE oss_load_static_part (
  c_int bigint,
  c_double double,
  c_string string,
  c_bool boolean,
  c_datetime string )
partitioned by (ds string);
-- Import data in open-source formats from OSS to oss_load_static_part.
LOAD overwrite table oss_load_static_part partition(ds='20190101')
from
LOCATION 'oss://<yourAccessKeyId>:<yourAccessKeySecret>@oss-cn-hangzhou-zmf.aliyuncs.com/my_bucket_id/my_location/'
STORED AS PARQUET;
```

 **Note** Data is loaded to a partitioned table. The schema of the data in the external storage path does not define partitioning columns. In this example, the external data only includes data in the `c_int`, `c_double`, `c_datetime`, `c_string`, and `c_bool` columns.

- Import data to a destination table in dynamic partition mode.

If the subdirectories of an OSS table directory are organized by partition name, you can import data to a partitioned table in dynamic partition mode.

```
-- The following code shows the organization of the OSS table directory. Data in the external storage path does not include data in partitioning columns. The information of partitioning columns is part of the path.
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/text_data/ds=20170102/'
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/text_data/ds=20170103/'

-- Create destination table oss_load_dyn_part.
CREATE TABLE oss_load_dyn_part (
  int bigint,
  c_double double,
  c_string string,
  c_bool boolean)
PARTITIONED BY (ds string);
Import data to oss_load_dyn_part in dynamic partition mode.
LOAD overwrite table oss_load_dyn_part partition(ds)
from
LOCATION 'oss://accessId:accessKey@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/text_data/'
ROW FORMAT serde 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES(
  'Fields terminator'='\001',
  'Escape delimiter'='\ ',
  'Collection items terminator'='\002',
  'Map keys terminator'='\003',
  'Lines terminator'='\n',
  'Null definition'='\N')
STORED AS TEXTFILE
```

In this example:

- `ROW FORMAT` and `SERDEPROPERTIES` can be omitted if you use their default values.

- You can set `SERDEPROPERTIES` for data of the TEXT type. The following code lists the default values:

```
Fields terminator:'\001'
Escape delimiter:'\'
Collection items terminator:'\002'
Map keys terminator:'\003'
Lines terminator:'\n'
Null defination:'\N'
```

3.13. DDL SQL

3.14. Insert Operation

3.15. Builtin functions

3.15.1. Comparison of functions built in MaxCompute, MySQL, and Oracle

This topic compares functions built in MaxCompute, MySQL, and Oracle.

comparison of built-in functions

Function type	MaxCompute	HIVE	MySQL	Oracle	Partition pruning supported in MaxCompute SQL
	DATEDIFF	DATEDIFF	DATEDIFF	MONTHS_BETWEEN	• Supported in MaxCompute mode • Not supported in Hive-compatible mode

	Date functions	DATE_ADD	DATE_ADD	N/A	- Not supported in MaxCompute mode. We recommend that you use the DATEADD function instead. - Supported in Hive-compatible mode
	DATEPART	N/A	DATE_FORMAT	EXTRACT (datetime)	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	DATETRUNC	TRUNC	DATE_FORMAT	EXTRACT (datetime)	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	FROM_UNIXTIME	FROM_UNIXTIME	FROM_UNIXTIME	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	GETDATE	CURRENT_DATE	NOW	CURRENT_DATE	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	ISDATE	N/A	STR_TO_DATE (If false is returned, the string cannot be converted to DATE.)	N/A	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	LASTDAY	LAST_DAY	LAST_DAY	LAST_DAY	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	TO_DATE	TO_DATE	STR_TO_DATE()	DATE	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	TO_CHAR	N/A	DATE_FORMAT	TO_CHAR (datetime)	- Supported in MaxCompute mode - Supported in Hive-compatible mode

Date functions	UNIX_TIMESTAMP	UNIX_TIMESTAMP	UNIX_TIMESTAMP	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	WEEKDAY	N/A	WEEKDAY	N/A	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	WEEKOFYEAR	WEEKOFYEAR	WEEKOFYEAR	N/A	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	YEAR	YEAR	YEAR	YEAR	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode

	QUARTER	QUARTER	QUARTER	QUARTER	• Not supported in MaxCompute mode • Not supported in Hive-compatible mode
	MONTH	MONTH	MONTH	MONTH	• Not supported in MaxCompute mode • Not supported in Hive-compatible mode
	DAY	DAY	DAY	DAY	• Not supported in MaxCompute mode • Not supported in Hive-compatible mode
	DAYOFMONTH	DAYOFMONTH	DAYOFMONTH	N/A	• Not supported in MaxCompute mode • Supported in Hive-compatible mode

	hour	hour	hour	hour	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode
	minute	minute	minute	minute	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode
	current_time stamp	current_time stamp	current_time stamp	current_time stamp	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode
	add_months	add_months	adddate	add_months	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode

	LAST_DAY	LAST_DAY	LAST_DAY	N/A	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode
	NEXT_DAY	NEXT_DAY	N/A	NEXT_DAY	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode
	MONTHS_BETWEEN	MONTHS_BETWEEN	timestampdiff	MONTHS_BETWEEN	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode
	ABS	ABS	ABS	ABS	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	ACOS	ACOS	ACOS	ACOS	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	ASIN	ASIN	ASIN	ASIN	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	ATAN	ATAN	ATAN	ATAN	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	CEIL	CEIL	CEIL	CEIL	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	CONV	CONV	CONV	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	COS	COS	COS	COS	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	COSH	COSH	N/A	COSH	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	COT	COT	COT	COT	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	EXP	EXP	EXP	EXP	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	FLOOR	FLOOR	FLOOR	FLOOR	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	LN	LN	LN	LN	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	LOG	LOG	LOG	LOG	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	POW	POW	POW	POWER	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	RAND	RAND	RAND	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	ROUND	ROUND	ROUND	ROUND	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	SIN	SIN	SIN	SIN	- Supported in MaxCompute mode - Supported in Hive-compatible mode

Mathematical functions	SINH	SINH	N/A	SINH	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	SQRT	SQRT	SQRT	SQRT	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	TAN	TAN	TAN	TAN	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	TANH	TANH	N/A	TANH	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	TRUNC	TRUNC	TRUNCATE	TRUNC	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	LOG2	LOG2	LOG2	LOG	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	LOG10	LOG10	LOG10	LOG	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	BIN	BIN	BIN	BITAND	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	HEX	HEX	HEX	RAWTOHEX	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	UNHEX	UNHEX	UNHEX	HEXTORAW	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	RADIANS	RADIANS	RADIANS	RADIANS	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	DEGREES	DEGREES	DEGREES	DEGREES	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	SIGN	SIGN	SIGN	SIGN	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	E	E	N/A	N/A	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	PI	PI	PI	PI	- Supported in MaxCompute mode - Not supported in Hive-compatible mode

	FACTORIAL	FACTORIAL	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	CBRT	CBRT	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	SHIFLEFT	SHIFLEFT	<<	N/A	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode
	SHIFTRIGHT	SHIFTRIGHT	>>	N/A	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode
	SHIFTRIGHTUNSIGNED	SHIFTRIGHTUNSIGNED	>>>	N/A	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode

Window functions	DENSE_RANK	DENSE_RANK	DENSE_RANK	DENSE_RANK	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	RANK	RANK	RANK	RANK	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	LAG	LAG	LAG	LAG	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	LEAD	LEAD	LEAD	LEAD	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	PERCENT_RANK	PERCENT_RANK	PERCENT_RANK	PERCENT_RANK	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	ROW_NUMBER	ROW_NUMBER	ROW_NUMBER	ROW_NUMBER	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	CLUSTER_SAMPLE	N/A	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	NTILE	NTILE	NTILE	NTILE	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	COUNT	COUNT	COUNT	COUNT	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	AVG	AVG	AVG	AVG	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	MAX	MAX	MAX	MAX	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	MIN	MIN	MIN	MIN	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	MEDIAN	N/A	N/A	MEDIAN	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	STDDEV	STDDEV	STDDEV	STDDEV	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	STDDEV_SAMP	STDDEV_SAMP	STDDEV_SAMP	STDDEV_SAMP	- Supported in MaxCompute mode - Supported in Hive-compatible mode

Aggregate functions	SUM	SUM	SUM	SUM	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	WM_CONCAT	N/A	GROUP_CONCAT	WM_CONCAT	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	COLLECT_LIST	COLLECT LIST	N/A	COLLECT	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	COLLECT_SET	COLLECT SET	N/A	COLLECT	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	VARIANCE/VAR_POP	VARIANCE/VAR_POP	VAR_POP	VARIANCE/VAR_POP	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	VAR_SAMP	VAR_SAMP	VAR_SAMP	VAR_SAMP	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	COVAR_POP	COVAR_POP	N/A	COVAR_POP	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	COVAR_SAMP	COVAR_SAMP	N/A	COVAR_SAMP	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	PERCENTILE	PERCENTILE	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	CHAR_MATCHCOUNT	N/A	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	CHR	CHR	CHAR	CHR	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	CONCAT	CONCAT	CONCAT	CONCAT	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	GET_JSON_OBJECT	GET_JSON_OBJECT	JSON_EXTRACT()	N/A	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	INSTR	INSTR	INSTR	INSTR	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	IS_ENCODING	N/A	N/A	N/A	- Supported in MaxCompute mode - Not supported in Hive-compatible mode

	KEYVALUE	N/A	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	LENGTH	LENGTH	LENGTH	LENGTH	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	LENGTHB	LENGTHB	LENGTHB	LENGTHB	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	MD5	MD5	MD5	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	REGEXP_EXTRACT	REGEXP_EXTRACT	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	REGEXP_INSTR	N/A	REGEXP_INSTR	REGEXP_INSTR	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	REGEXP_REPLACE	REGEXP_REPLACE	REGEXP_REPLACE	REGEXP_REPLACE	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	REGEXP_SUBSTR	N/A	REGEXP_SUBSTR	REGEXP_SUBSTR	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	REGEXP_COUNT	N/A	N/A	REGEXP_COUNT	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	SPLIT_PART	N/A	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode

String functions	SUBSTR	SUBSTR	SUBSTR	SUBSTR	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	SUBSTRING	SUBSTRING	SUBSTRING	SUBSTR	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	TOLOWER	LOWER	LOWER	LOWER	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	TOUPPER	UPPER	UPPER	UPPER	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	TRIM	TRIM	TRIM	TRIM	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	LTRIM	LTRIM	LTRIM	LTRIM	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	RTRIM	RTRIM	RTRIM	LTRIM	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	REVERSE	REVERSE	REVERSE	REVERSE	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	SPACE	SPACE	SPACE	SPACE	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	REPEAT	REPEAT	REPEAT	REPEAT	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	ASCII	ASCII	ASCII	ASCII	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	CONCAT_WS	CONCAT_WS	CONCAT_WS	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	LPAD	LPAD	LPAD	LPAD	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	RPAD	RPAD	RPAD	RPAD	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	REPLACE	N/A	REPLACE	REPLACE	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	SOUNDEX	SOUNDEX	SOUNDEX	SOUNDEX	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	SUBSTRING_INDEX	SUBSTRING_INDEX	SUBSTRING_INDEX	N/A	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode
	TRANSLATE	TRANSLATE	N/A	TRANSLATE	- Not supported in MaxCompute mode - Not supported in Hive-compatible mode
	URL_DECODE	N/A	N/A	N/A	- Supported in MaxCompute mode - Not supported in Hive-compatible mode

	URL_ENCODE	N/A	N/A	PERCENTILE_CONT	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	CRC32	CRC32	CRC32	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	CAST	CAST	CAST	CAST	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	COALESCE	COALESCE	COALESCE	COALESCE	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	DECODE	DECODE	N/A	DECODE	- Supported in MaxCompute mode - Not supported in Hive-compatible mode

	GET_IDCARD_AGE	N/A	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	GET_IDCARD_BIRTHDAY	N/A	N/A	N/A	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	GET_IDCARD_SEX	N/A	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	GREATEST	GREATEST	GREATEST	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	ORDINAL	N/A	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	LEAST	LEAST	LEAST	LEAST	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	MAX_PT	N/A	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	UUID	N/A	UUID	UID	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	SAMPLE	N/A	N/A	N/A	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	IF	IF	IF	IF	- Supported in MaxCompute mode - Supported in Hive-compatible mode

Other functions	CASE WHEN	CASE WHEN	CASE WHEN	CASE WHEN	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	SPLIT	SPLIT	SPLIT	N/A	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	STR_TO_MAP	STR_TO_MAP	N/A	N/A	- Supported in MaxCompute mode - Not supported in Hive-compatible mode
	EXPLODE	EXPLODE	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	MAP	MAP	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	MAP_KEYS	MAP_KEYS	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	MAP_VALUES	MAP_VALUES	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	NVL	NVL	IFNULL	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	ARRAY	ARRAY	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	SIZE	SIZE	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode

	ARRAY_CONTAINS	ARRAY_CONTAINS	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	POSEXplode	POSEXplode	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	TRANS_ARRAY	N/A	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	INLINE	INLINE	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode
	NAMED_STRUCT	N/A	N/A	N/A	- Supported in MaxCompute mode - Supported in Hive-compatible mode

 **Note** The MaxCompute mode is enabled by default. To use the Hive-compatible mode, run the following commands:

```
-- Switch to the Hive-compatible mode at the project level.  
setproject odps.sql.hive.compatible=True;  
  
-- Switch to the Hive-compatible mode at the session level.  
set odps.sql.hive.compatible=True;
```

3.15.2. Date functions

This article explains various functions that MaxCompute SQL offers to operate datetime types.

DATEADD

Command format:

```
datetime dateadd(datetime date, bigint delta, string datepart)
```

Command description:

Modify the value of date according to a specified unit 'datepart' and specified scope 'delta'.

Parameter description:

- **date**: Datetime type, value of date. If the input is string type, it is converted to 'datetime' type by implicit conversion. If it is another type, an exception is indicated.
- **delta**: Bigint type, date scope to be modified. If the input is 'string' type or 'double' type, it is converted to 'bigint' type by implicit conversion. If it is another data type, exception occurs. If 'delta' is greater than zero, do 'add' operation, otherwise do 'minus' operation.
- **datepart**: a String type constant. This field value follows 'string' and 'datetime' type conversion agreement, where, 'yyyy' indicates year; 'mm' indicates month.

See Conversion between [String type and Datetime type](#). In addition, the extensional date format is also supported: year- 'year'; month- 'month' or 'mon'; day- 'day'; hour- 'hour'. If it is not a constant or unsupported format or other data type, an exception is indicated.

Return value: Datetime type. If any input is NULL, return NULL.

 Note

- While increasing or decreasing 'delta' according to specified unit, it causes the carry or back space for higher unit. Day, month, hour, minute, second are calculated by 10 hexadecimal, 12 hexadecimal, 24 hexadecimal, 60 hexadecimal, 60 hexadecimal respectively.
- If the unit of 'delta' is month, the calculation rule is shown as follows:
If the month part of 'datetime' does not cause the spillover of day after adding 'delta', then do not change the day, else the day value is set to the last day of the result month.
- The value of 'datepart' follows 'string' and 'datetime' type conversion agreement, that is, 'yyyy' indicates year; 'mm' indicates month and so on. If no special description exists, related datetime built-in functions follow this agreement. Moreover, if no special instructions, the part of all datetime built-in functions supports extended date format: year- 'year'; month-'month' or 'mon'; day-'day'; hour-'hour'.

For example:

```
select dateadd(datetime '2005-02-28 00:00:00', 1, 'dd') ;
--return 2005-03-01 00:00:00.Add one day. The result is beyond the last day in February. The actual value is the first day of next month.
select dateadd(datetime '2005-02-28 00:00:00', -1, 'dd');
--return 2005-02-27 00:00:00.Minus one day.
select dateadd(datetime '2005-02-28 00:00:00', 20, 'mm');
--return 2006-10-28 00:00:00.Add 20 months. The month spillover is caused and the year is added '1'.
select dateadd(datetime '2005-02-28 00:00:00', 1, 'mm');
--return 2005-03-28 00:00:00
select dateadd(datetime '2005-01-29 00:00:00', 1, 'mm');
--return 2005-02-28 00:00:00.No 29th is in Feb. of 2005. The date is intercepted to the last day of current month.
select dateadd(datetime '2005-03-30 00:00:00', -1, 'mm');
--return 2005-02-28 00:00:00
```

In MaxCompute SQL, the datetime type has no direct constant representation, the following usage is wrong:

```
select dateadd(2005-03-30 00:00:00, -1, 'mm') from tbl1;
```

If you must describe the datetime type constant, try the following methods:

```
select dateadd(cast("2005-03-30 00:00:00" as datetime), -1, 'mm') from tbl1;
-- The String type constant is converted to datetime type by explicit conversion.
```

DATEDIFF

Command format:

```
bigint datediff(datetime date1, datetime date2, string datepart)
```

Command description:

Calculate the difference between two datetime date1 and date2 in specified time unit 'datepart'.

Parameter description:

- date1, date2: Datetime type, minuend, meiosis. If the input is 'string', it is converted to 'datetime' by implicit conversion. If it is another data type, an exception is indicated.
- datepart: a String type constant. The extensional date format is supported. If 'datepart' does not meet the specified format or is other data type, an exception is indicated.

Return value:

Returns the Bigint type. Any input parameter is NULL, return NULL. If date1 is less than date2, then the returned value may be negative.

 **Note** The lower unit part is cut off according to 'datepart' in the calculation process and then calculate the result.

For example:

If start = 2005-12-31 23:59:59, end = 2006-01-01 00:00:00:

```
datediff(end, start, 'dd') = 1
```

```
datediff(end, start, 'mm') = 1
```

```
datediff(end, start, 'yyyy') = 1
```

```
datediff(end, start, 'hh') = 1
```

```
datediff(end, start, 'mi') = 1
```

```
datediff(end, start, 'ss') = 1
```

```
datediff(datetime '2013-05-31 13:00:00', '2013-05-31 12:30:00', 'ss') = 1800
```

```
datediff(datetime '2013-05-31 13:00:00', '2013-05-31 12:30:00', 'mi') = 30
```

If start = 19:33:23. 234, end = 19:33:23. 250 .Dates with milliseconds do not belong to the standard datetime style, and cannot be converted implicitly directly. Explicit conversion is required here:

```
datediff(to_date('2018-06-04 19:33:23.250', 'yyyy-MM-dd hh:mi:ss.ff3'), to_date('2018-06-04 19:33:23.234', 'yyyy-MM-dd hh:mi:ss.ff3'), 'ff3') = 16
```

DATEPART**Command format:**

```
bigint datepart(datetime date, string datepart)
```

Command format:

Extracts the value of the specified time unit 'datepart' in 'date'.

Parameter description:

Return value:

- **date:** Datetime type. If the input is 'string' type, it is converted to 'datetime' type. If it is another data type, an exception is indicated.
- **datepart:** String type constant. The extensional date format is supported. If 'datepart' does not meet the specified format or is other data type, an exception is indicated.

Returns the Bigint type. If any input is NULL, return NULL.

For example:

```
datepart(datetime '2013-06-08 01:10:00', 'yyyy') = 2013
datepart(datetime '2013-06-08 01:10:00', 'mm') = 6
```

DATETRUNC

Command format:

```
datetime datetrunc (datetime date, string datepart)
```

Usage::

Return the remained date value after the specified time unit 'datepart' has been intercepted.

Parameter description::

- **date:** Datetime type. If the input is 'string' type, it is converted to 'datetime' type. If it is another data type, an exception is indicated.
- **datepart:** String type constant. The extensional date format is supported. If 'datepart' does not meet the specified format or is other data type, an exception is indicated.

Return value:

Datetime type. If any input is NULL, return NULL.

For example:

```
datetrunc(datetime '2011-12-07 16:28:46', 'yyyy') = 2011-01-01 00:00:00
datetrunc(datetime '2011-12-07 16:28:46', 'month') = 2011-12-01 00:00:00
datetrunc(datetime '2011-12-07 16:28:46', 'DD') = 2011-12-07 00:00:00
```

GETDATE

Command format:

```
datetime getdate()
```

Command description:

Get present system time. Use UTC+8 as MaxCompute standard time.

Return value:

Datetime type, return present date and time.

 **Note** In a MaxCompute SQL task (executed in a distributed manner), 'getdate' always returns a fixed value. The return result is any time in MaxCompute SQL execution period and the precision of time is accurate to seconds.

ISDATE

Command format:

```
boolean isdate(string date, string format)
```

Command description:

Determines whether a date string can be converted to a datetime value according to corresponding format string. If the conversion is successful, return TRUE, otherwise return FALSE.

Parameter description:

- **date**: date value of String format. If the input is 'bigint', or 'double' or 'datetime', it is be converted to 'string' type. If it is another data type, an exception is indicated.
- **format**: a String type constant. The extensional date format is not supported. If redundant format strings appear in 'format', then get the datetime value corresponding to the first format string, other strings are taken as separators. For example, isdate ('1234-yyyy', 'yyyy-yyyy') returns 'TRUE'.

Return value:

Boolean type. If any parameter is NULL, return NULL.

LASTDAY

Command format:

```
datetime lastday(datetime date)
```

Command format:

Get the last day in the same month of the date, intercepted to day and the 'hh:mm:ss' part is '00:00:00'.

Parameter description:

date: Datetime type. If the input is 'string' type, it is converted to 'datetime' type. If it is another data type, an exception is reported.

Return value:

Datetime type. If the input is NULL, return NULL.

TO_DATE

Command format:

```
datetime to_date(string date, string format)
```

Command description:

Convert a string 'date' to the datetime value according to a specified format.

Parameter description:

- **date:** String type, date value to be converted. If the input is 'bigint', or 'double' or 'datetime', it is converted to 'string' type by implicit conversion. If it is another data type or null, an exception is indicated.
- **format:** String type constant, date format. If it is not a constant or is other data type, the exception is caused. The field 'format' does not support extensional format and other characters are ignored as invalid characters in analysis process.

The parameter format contains 'yyyy' at least; otherwise the exception is indicated. If redundant format strings appear in format, then get the datetime value corresponding to the first format string, other strings are taken as separators. For example, `to_date('1234-2234', 'yyy y-yyyy')` returns '1234-01-01 00:00:00'.

Format format: yyyy is a four-digit year, mm is a two-digit month, DD is a two-digit day, HH is a 24-hour system, MI is a two-digit minute, SS is a two-digit second, FF3 is a three-digit precision millisecond.

Return value:

Datetime type, the format is yyyy-mm-dd hh: mi: ss. If any input is NULL, return NULL.

For example:

```
to_date('Alibaba2010-12*03', 'Alibabayyy-mm*dd') = 2010-12-03 00:00:00
to_date('20080718', 'yyyymmdd') = 2008-07-18 00:00:00
to_date('200807182030', 'yyyymmddhhmi')=2008-07-18 20:30:00
to_date('2008718', 'yyyymmdd') = null
-- The format does not meet the requirements. An exception is thrown.
to_date('Alibaba2010-12*3', 'Alibabayyy-mm*dd') = null
-- Format is not compatible and exception is thrown.
to_date('2010-24-01', 'yyyy') = null
-- Format is not compatible and exception is thrown.
to_date('20181030 15-13-12.345', 'yyyymmdd hh-mi-ss.ff3')=2018-10-30 15:13:12
```

TO_CHAR

Command format:

```
string to_char(datetime date, string format)
```

Command description:

Convert the 'date' of datetime type to a string according to a specified format.

Parameter description:

- **date:** Datetime type, the date value to be converted. If the input is 'string' type, it is converted to 'datetime' type by implicit conversion. If it is another data type, an exception indicated.
- **format:** String type constant. If it is not a constant or is other data type, the exception is indicated. In 'format', the date format part is replaced with the corresponding data and other characters are output directly.

Return value:

Returns the String type. Any input parameter is NULL, return NULL.

For example:

```
to_char(datetime '2010-12-03 00:00:00', 'Alibabayyyy-mm*dd') = 'Alibaba2010-12*03'
to_char(datetime '2008-07-18 00:00:00', 'yyyymmdd') = '20080718'
to_char(datetime 'Alibaba2010-12*3', 'Alibabayyyy-mm*dd') -- Format is not compatible and exception is thrown.
to_char(datetime '2010-24-01', 'yyyy') -- Format is not compatible and exception is thrown.
to_char(datetime '2008718', 'yyyymmdd') -- Format is not compatible and exception is thrown.
```

See [TO_CHAR](#) for conversion from other types to string type.

UNIX_TIMESTAMP**Command format:**

```
bigint unix_timestamp(datetime date)
```

Command description:

Convert the date of Datetime type to UNIX format date of Bigint type.

Parameter description:

date: Datetime type date value. If the input is 'string' type, it is converted to 'datetime' type and involved in calculation. If it is another type, an exception indicated.

Return value:

Bigint type, it indicates UNIX format date value. If 'date' is NULL, return NULL.

for example:

```
select unix_timestamp(datetime '2009-03-20 11:11:00'); --return 1237518660.
```

FROM_UNIXTIME**Command format:**

```
datetime from_unixtime(bigint unixtime)
```

Command description:

Convert the numeric UNIX time value 'unixtime' to datetime value.

Parameter description:

unixtime: Bigint type, number of seconds, UNIX format date time value. If the input is 'string', 'double', it is converted to 'bigint' type by implicit conversion.

Return value:

Datetime type date value. If 'unixtime' is NULL, return NULL.

 **Note** If you have `set odps.sql.hive.compatible=true;`, and the input type is string, the return type will be string too.

For example:

```
from_unixtime(123456789) = 1973-11-30 05:33:09
```

WEEKDAY

Command format:

```
bigint weekday(datetime date)
```

Command description:

Return the nth day of present week corresponding to the date.

Parameter description:

date: Datetime type. If the input is 'string' type, it is converted to 'datetime' type and then involved in operation. If it is another date type, an exception indicated.

Return value:

Bigint type. If the input parameter is NULL, return NULL. Monday is the first day of a week and the return value is 0. Days are in ascending order starting from 0. If the day is Sunday, then return is 6.

WEEKOFYEAR

Command format:

```
bigint weekofyear(datetime date)
```

Command description:

Return the nth week of a year which the date is included in. Monday is taken as the first day of a week.

 **Note** Whether this week belongs to this year, or the next year, it depends on which year (4 days or more) most of the time of this week belongs to.

Parameter description:

date: Datetime type. If the input is 'string' type, it is converted to 'datetime' type and then involved in operation. If it is another date type, an exception is indicated.

Return value:

Bigint type. If the input is NULL, return NULL.

For example:

```
select weekofyear(to_date("20141229", "yyyymmdd")) from dual;
```

Result:

```
+-----+
```

```
|_c0|
```

```
+-----+
```

```
| 1 |
```

```
+-----+
```

-Although 20141229 belongs to 2014, most of the dates of the week are in 2015, therefore, the return result is 1, indicating that it is the first week of 2015.

```
select weekofyear(to_date("20141231", "yyyymmdd")) from dual;
```

```
-- Return 1.
```

```
select weekofyear(to_date("20141229", "yyyymmdd")) from dual;
```

```
-- Return 53.
```

Maxcomputerte2.0 New Extended Mathematical Functions

With the upgraded version of MaxCompute 2.0, some new date functions are added to the product. If the functions are used to design a new data type compatible with the Hive mode, you must add the following two set statements before the SQL statement of the new functions:

```
set odps.sql.type.system.odps2=true;--Enable the new type.
```

If you want to submit both at the same time, run the following statements:

```
set odps.sql.type.system.odps2=true;
```

```
select year('1970-01-01 12:30:00');--return 1970.
```

The new extended functions are described as follows.

YEAR

Command format:

```
INT year(string date)
```

Command description:

Returns the year of a date.

Parameter description:

date: String-type date value. The format must at least include 'yyyy-mm-dd' and cannot include additional strings. Otherwise, null is returned.

Return value:

INT type.

For example:

```
year('1970-01-01 12:30:00') = 1970
year('1970-01-01') = 1970
year('70-01-01') = 70
year(1970-01-01) = 1970
year('1970/03/09') = null
year(null) Returns an exception
```

QUARTER

Command format:

```
INT quarter(datetime/timestamp/string date )
```

 **Note** Before the SQL statement which uses the QUARTER function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the quarter of a date, range: 1-4.

Parameter description:

date: Datetime, Timestamp, or String-type date value. The format must at least include 'yyyy-mm-dd'. Otherwise, null is returned.

Return value:

Int type, null input returns null.

For example:

```
quarter('1970-11-12 10:00:00') = 4
quarter('1970-11-12') = 4
```

MONTH

Command format:

```
INT month(string date)
```

 **Note** Before the SQL statement which uses the MONTH function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the month of a date.

Parameter description:

date: String-type date value. Other value types return an exception.

Return value:

INT type.

For example:

```
month('2014-09-01') = 9
month('20140901') = null
```

DAY

Command format:

```
INT day(string date)
```

 **Note** Before the SQL statement which uses the function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the day of a date.

Parameter description:

date: String-type date value. Other value types return an exception.

Return value:

INT type.

For example:

```
day('2014-09-01') = 1
day('20140901') = null
```

DAYOFMONTH

Command format:

```
INT dayofmonth(date)
```

 **Note** Before the SQL statement which uses the DAYOFMONTH function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the day of a date.

For example, after command `int dayofmonth(2017-10-13)` runs, 13 returns.

Parameter description:

date: String-type date value. Other value types return an exception.

Return value:

INT type.

For example:

```
dayofmonth('2014-09-01') = 1  
dayofmonth('20140901') = null
```

HOUR

Command format:

```
INT hour(string date)
```

 **Note** Before the SQL statement which uses the HOUR function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the hour of a date.

Parameter description:

date: String-type date value. Other value types return an exception.

Return value:

Int type.

For example:

```
hour('2014-09-01 12:00:00')=12  
hour('12:00:00')=12  
hour('20140901120000')=null
```

MINUTE

Command format:

```
INT minute(string date)
```

 **Note** Before the SQL statement which uses the MINUTE function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the minute of a date.

Parameter description:

date: String-type date value. Other value types return an exception.

Return value:

Int type.

For example:

```
minute('2014-09-01 12:30:00') = 30
minute('12:30:00') = 30
minute('20140901120000') = null
```

SECOND

Command format:

```
INT second(string date)
```

 **Note** Before the SQL statement which uses the SECOND function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the second of a date.

Parameter description:

date: String-type date value. Other value types return an exception.

Return value:

INT type.

For example:

```
second('2014-09-01 12:30:45') = 45
second('12:30:45') = 45
second('20140901123045') = null
```

CURRENT_TIMESTAMP

Command format:

```
timestamp current_timestamp()
```

 **Note** Before the SQL statement which uses the CURRENT_TIMESTAMP function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the current timestamp as a Timestamp-type value. The value is not fixed.

Return value:

Timestamp type.

For example:

```
select current_timestamp() from dual;--Returns '2017-08-03 11:50:30.661'
```

ADD_MONTHS

Command format:

```
string add_months(string startdate, int nummonths)
```

 **Note** Before the SQL statement which uses the ADD_MONTHS function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the date given by startdate plus the nummonths value.

Parameter description:

- **startdate:** String-type value. The format must at least include the 'yyyy-mm-dd' date. Otherwise, null is returned.
- **num_months:** Int-type value.

Return value:

A String-type date, in the format 'yyyy-mm-dd'.

For example:

```
Add_months ('2017-02-14', 3) = '2017-05-14'  
add_months('17-2-14',3) = '0017-05-14'  
add_months('2017-02-14 21:30:00',3) = '2017-05-14'  
add_months('20170214',3) = null
```

LAST_DAY

Command format:

```
string last_day(string date)
```

 **Note** Before the SQL statement which uses the LAST_DAY function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the date of the last day of the month that contains the given date.

Parameter description:

date: String type, with the format 'yyyy-MM-dd HH:mi:ss' or 'yyyy-MM-dd'.

Return value:

A String-type date, in the format 'yyyy-mm-dd'.

For example:

```
last_day('2017-03-04') = '2017-03-31'
last_day('2017-07-04 11:40:00') = '2017-07-31'
last_day('20170304') = null
```

NEXT_DAY

Command format:

```
string next_day(string startdate, string week)
```

 **Note** Before the SQL statement which uses the NEXT_DAY function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the first date larger than the specified startdate that matches the day of the week given by the week parameter. It is the date of a specific day in the next week.

Parameter description:

- **startdate:** String type, with the format 'yyyy-MM-dd HH:mi:ss' or 'yyyy-MM-dd'.
- **week:** String type, the first two or three letters of a day of the week, or the full name of the day of the week. For example: Mo, TUE, or FRIDAY.

Return value:

A String-type date, in the format 'yyyy-mm-dd'.

For example:

```
next_day('2017-08-01','TU') = '2017-08-08'
next_day('2017-08-01 23:34:00','TU') = '2017-08-08'
Next_day ('20170801 ', 'tu') = NULL
```

MONTHS_BETWEEN

Command format:

```
double months_between(datetime/timestamp/string date1, datetime/timestamp/string date2)
```

 **Note** Before the SQL statement which uses the MONTHS_BETWEEN function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Command description:

Returns the number of months between date1 and date2.

Parameter description:

- date1: Datetime, Timestamp, or String type, with the format 'yyyy-MM-dd HH:mi:ss' or 'yyyy-MM-dd'.
- date2: Datetime, Timestamp, or String type, with the format 'yyyy-MM-dd HH:mi:ss' or 'yyyy-MM-dd'.

Return Value:

Returns the Double type.

- When date1 is later than date2, the returned value is positive. When date2 is later than date1, the returned value is negative.
- When date1 and date2 correspond to the last days of two months, the returned value is an integer representing the number of months. Otherwise, the formula is (date1 - date2)/31.

Examples:

```
months_between('1997-02-28 10:30:00', '1996-10-30') = 3.9495967741935485
months_between('1996-10-30','1997-02-28 10:30:00') = -3.9495967741935485
months_between('1996-09-30','1996-12-31') = -3.0
```

3.15.3. Mathematical functions

This topic describes the mathematical functions supported by MaxCompute SQL.

mathematical function

ABS

- Syntax

```
Double abs(Double number)
Bigint abs(Bigint number)
Decimal abs(Decimal number)
```

- Description

This function calculates the absolute value of number.

- Parameters

number: If number is of the DOUBLE, BIGINT, or DECIMAL type, a value of the same type is returned.

- If number is of the BIGINT type, the return value must be of the BIGINT type.
- If number is of the DOUBLE type, the return value must be of the DOUBLE type.
- If number is of the DECIMAL type, the return value must be of the DECIMAL type.
- If number is of the STRING type, it is implicitly converted to a value of the DOUBLE type before calculation.
- If number is of a data type other than the preceding four types, an error is returned.

 **Note** If number of the BIGINT type exceeds the upper limit, a value of the DOUBLE type is returned. However, the precision may be lost.

- Return value

The type of the return value depends on that of the input value, which can be DOUBLE, BIGINT, or DECIMAL. If the input value is NULL, NULL is returned.

- Examples

```
abs(null)=null
abs(-1)=1
abs(-1.2)=1.2
abs("-2")=2.0
abs(122320837456298376592387456923748)=1.2232083745629837e32
```

The following example shows the usage of a complete ABS function in SQL statements. Other built-in functions, except window functions and aggregation functions, are used in a similar way.

```
-- Calculate the absolute value of the id field in tbl1.
select abs(id) from tbl1;
```

ACOS

- Syntax

```
Double acos(Double number)
Decimal acos(Decimal number)
```

- Description

This function calculates the arccosine of number.

- Parameters

number: a value of the DOUBLE or DECIMAL type. The value ranges from -1 to 1. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- Return value

A value of the DOUBLE or DECIMAL type is returned. The value ranges from 0 to π . If the input value is NULL, NULL is returned.

- Examples

```
acos("0.87")=0.5155940062460905  
acos(0)=1.5707963267948966
```

ASIN

- Syntax

```
Double asin(Double number)  
Decimal asin(Decimal number)
```

- Description

This function calculates the arcsine of number.

- Parameters

number: a value of the DOUBLE or DECIMAL type. The value ranges from -1 to 1. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- Return value

A value of the DOUBLE or DECIMAL type is returned. The value ranges from $-\pi/2$ to $\pi/2$. If the input value is NULL, NULL is returned.

- Examples

```
asin(1)=1.5707963267948966  
asin(-1)=-1.5707963267948966
```

ATAN

- Syntax

```
Double atan(Double number)
```

- Description

This function calculates the arctangent of number.

- Parameters

number: a value of the DOUBLE type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- Return value

A value of the DOUBLE type is returned. The value ranges from $-\pi/2$ to $\pi/2$. If the input value is NULL, NULL is returned.

- Examples

```
atan(1)=0.7853981633974483
atan(-1)=-0.7853981633974483
```

CEIL

- Syntax

```
Bigint ceil(Double value)
Bigint ceil(Decimal value)
```

- Description

This function rounds up value and returns the nearest integer.

- Parameters

value: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- Return value

A value of the BIGINT type is returned. If the input value is NULL, NULL is returned.

- Examples

```
ceil(1.1)=2
ceil(-1.1)=-1
```

CONV

- Syntax

```
String conv(String input, Bigint from_base, Bigint to_base)
```

- Description

This function converts a number from one number system to another.

- Parameters

- input: the integer you want to convert, which is of the STRING type. Implicit conversions between the BIGINT and DOUBLE types are supported.
- from_base and to_base: decimal numbers. The values can be 2, 8, 10, or 16. Implicit conversions between the STRING and DOUBLE types are supported.

- Return value

A value of the **STRING** type is returned. If any input value is **NULL**, **NULL** is returned. The conversion process runs at 64-bit precision. An error is returned if an overflow occurs. If the input value is a negative value, which begins with an en dash (-), an error is returned. If the input value is a decimal, it is converted to an integer before the conversion of number systems. The decimal part is left out.

- Examples

```
conv('1100', 2, 10)='12'  
conv('1100', 2, 16)='c'  
conv('ab', 16, 10)='171'  
conv('ab', 16, 16)='ab'
```

COS

- Syntax

```
Double cos(Double number)  
Decimal cos(Decimal number)
```

- Description

This function calculates the cosine of number, which is a radian value.

- Parameters

number: a value of the **DOUBLE** or **DECIMAL** type. If the input value is of the **STRING** or **BIGINT** type, it is implicitly converted to a value of the **DOUBLE** type before calculation. If the input value is of another data type, an error is returned.

- Return value

A value of the **DOUBLE** or **DECIMAL** type is returned. If the input value is **NULL**, **NULL** is returned.

- Examples

```
cos(3.1415926/2)=2.6794896585028633e-8  
cos(3.1415926)=-0.9999999999999986
```

COSH

- Syntax

```
Double cosh(Double number)  
Decimal cosh(Decimal number)
```

- Description

This function calculates the hyperbolic cosine of number.

- Parameters

number: a value of the **DOUBLE** or **DECIMAL** type. If the input value is of the **STRING** or **BIGINT** type, it is implicitly converted to a value of the **DOUBLE** type before calculation. If the input value is of another data type, an error is returned.

- Return value

A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

COT

- Syntax

```
Double cot(Double number)
Decimal cot(Decimal number)
```

- Description

This function calculates the cotangent of number, which is a radian value.

- Parameters

number: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- Return value

A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

EXP

- Syntax

```
Double exp(Double number)
Decimal exp(Decimal number)
```

- Description

This function calculates the exponential value of number.

- Return value

The exponential value of number is returned.

- Parameters

number: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- Return value

A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

FLOOR

- Syntax

```
Bigint floor(Double number)
Bigint floor(Decimal number)
```

- Description

This function rounds down number and returns the nearest integer.

- Parameters

number: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- Return value

A value of the BIGINT type is returned. If the input value is NULL, NULL is returned.

- Examples

```
floor(1.2)=1
floor(1.9)=1
floor(0.1)=0
floor(-1.2)=-2
floor(-0.1)=-1
floor(0.0)=0
floor(-0.0)=0
```

LN

- Syntax

```
Double ln(Double number)
Decimal ln(Decimal number)
```

- Description

This function returns the natural logarithm of number.

- Parameters

number: a value of the DOUBLE or DECIMAL type.

- If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.
- If the input value is NULL, NULL is returned. If the input value is negative or 0, an error is returned.

- Return value

A value of the DOUBLE or DECIMAL type is returned.

LOG

- Syntax

```
Double log(Double base, Double x)
Decimal log(Decimal base, Decimal x)
```

- Description

This function returns the logarithm of x whose base number is base.

- Parameters

- **base**: the base value, which is of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.
- **x**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.
- **Return value**

The logarithm value of the DOUBLE or DECIMAL type is returned.

 - If any input value is NULL, NULL is returned.
 - If any input value is negative or 0, an error is returned.
 - If the value of base is 1, which causes division by 0, an error is returned.

POW

- **Syntax**

```
Double pow(Double x, Double y)
Decimal pow(Decimal x, Decimal y)
```

- **Description**

This function returns the yth power of x, that is, x^y .

- **Parameters**

- **x**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.
- **y**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- **Return value**

A value of the DOUBLE or DECIMAL type is returned. If any input value is NULL, NULL is returned.

RAND

- **Syntax**

```
Double rand(Bigint seed)
```

- **Description**

This function returns a random number of the DOUBLE type with the seed specified by seed. The value ranges from 0 to 1.

- **Parameters**

seed: the random seed of the BIGINT type. This parameter is optional and specifies the starting point in generating random numbers.

- **Return value**

A value of the DOUBLE type is returned.

- Examples

```
select rand() from dual;  
select rand(1) from dual;
```

ROUND

- Syntax

```
Double round(Double number, [Bigint Decimal_places])  
Decimal round(Decimal number, [Bigint Decimal_places])
```

- Description

This function returns a number rounded to the specified decimal place.

- Parameters

- **number**: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.
- **Decimal_places**: the decimal place, which is a constant of the BIGINT type. The value is rounded to the decimal point. If it is of another type, an error is returned. If this parameter is not specified, the number is rounded to the ones place. The default value is 0.

 **Note** **Decimal_places** can be a negative value. The negative value is counted from the decimal point to the left and the decimal part is excluded. If **Decimal_places** exceeds the length of the integer part, 0 is returned.

- Return value

A value of the DOUBLE or DECIMAL type is returned. If any input value is NULL, NULL is returned.

- Examples

```
round(125.315)=125.0  
round(125.315, 0)=125.0  
round(125.315, 1)=125.3  
round(125.315, 2)=125.32  
round(125.315, 3)=125.315  
round(-125.315, 2)=-125.32  
round(123.345, -2)=100.0  
round(null)=null  
round(123.345, 4)=123.345  
round(123.345, -4)=0.0
```

SIN

- Syntax

```
Double sin(Double number)
Decimal sin(Decimal number)
```

- **Description**

This function calculates the sine of number, which is a radian value.

- **Parameters**

number: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- **Return value**

A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

SINH

- **Syntax**

```
Double sinh(Double number)
decimal sinh(decimal number)
```

- **Description**

This function calculates the hyperbolic sine of number.

- **Parameters**

number: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- **Return value**

A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

SQRT

- **Syntax**

```
Double sqrt(Double number)
Decimal sqrt(Decimal number)
```

- **Description**

This function calculates the square root of number.

- **Parameters**

number: a value of the DOUBLE or DECIMAL type. It must be greater than 0. If it is less than 0, an error is returned. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- **Return value**

A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

TAN

- Description

```
Double tan(Double number)
Decimal tan(Decimal number)
```

- Description

This function calculates the tangent of number, which is a radian value.

- Parameters

number: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- Return value

A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

TANH

- Syntax

```
Double tanh(Double number)
Decimal tanh(Decimal number)
```

- Description

This function calculates the hyperbolic tangent of number.

- Parameters

number: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- Return value

A value of the DOUBLE or DECIMAL type is returned. If the input value is NULL, NULL is returned.

TRUNC

- Syntax

```
Double trunc(Double number[, Bigint Decimal_places])
Decimal trunc(Decimal number[, Bigint Decimal_places])
```

- Description

This function truncates the input value of number and right-fills it with zeros from the specified position.

- Parameters

- number: a value of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to a value of the DOUBLE type before calculation. If the input value is of another data type, an error is returned.

- **Decimal_places**: the decimal place, which is a constant of the BIGINT type. This parameter indicates the position where the number is truncated. If it is of another data type, it is converted to the BIGINT type. If this parameter is not specified, the number is truncated to the ones place.

 **Note** **Decimal_places** can be a negative value, which indicates that the number is truncated from the decimal point to the left and the decimal part is left out. If **Decimal_places** exceeds the length of the integer part, 0 is returned.

- **Return value**

A value of the DOUBLE or DECIMAL type is returned. If any input value is NULL, NULL is returned.

 Note

- If a value of the DOUBLE type is returned, the return value may not be displayed properly. This issue exists in all systems. For more information, see `trunc(125.815,1)` in this example.
- The number is filled with zeros from the specified position.

- **Examples**

```
trunc(125.815)=125.0
trunc(125.815,0)=125.0
trunc(125.815,1)=125.8000000000000001
trunc(125.815,2)=125.81
trunc(125.815,3)=125.815
trunc(-125.815,2)=-125.81
trunc(125.815,-1)=120.0
trunc(125.815,-2)=100.0
trunc(125.815,-3)=0.0
trunc(123.345,4)=123.345
trunc(123.345,-4)=0.0
```

Additional functions of MaxCompute V2.0

If a new data type, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, needs to be used in a MaxCompute SQL statement, you must insert a SET command before the SQL statement to enable the new data type.

- **Session level**: To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, plus commit and execute them together.
- **Project level**: You can enable a new data type at the project level. The project owner can run the following command to configure a project:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Other operations](#). For the precautions you must

take when you enable data types at the project level, see [Date types](#).

LOG2

- Syntax

```
Double log2(Double number)
Double log2(Decimal number)
```

- Description

This function calculates the logarithm of number with the base number of 2.

- Parameters

number: a value of the DOUBLE or DECIMAL type.

- Return value

A value of the DOUBLE type is returned. If the input value is 0 or NULL, NULL is returned.

- Examples

```
log2(null)=null
log2(0)=null
log2(8)=3.0
```

LOG10

- Syntax

```
Double log10(Double number)
Double log10(Decimal number)
```

- Description

This function calculates the logarithm of number with the base number of 10.

- Parameters

number: a value of the DOUBLE or DECIMAL type.

- Return value

A value of the DOUBLE type is returned. If the input value is 0 or NULL, NULL is returned.

- Examples

```
log10(null)=null
log10(0)=null
log10(8)=0.9030899869919435
```

BIN

- Syntax

```
String bin(Bigint number)
```

- Description

This function calculates the binary code of number.

- Parameters

number: a value of the BIGINT type.

- Return value

A value of the STRING type is returned. If the input value is 0, 0 is returned. If the input value is NULL, NULL is returned.

- Examples

```
bin(0)='0'  
bin(null)='null'  
bin(12)='1100'
```

HEX

- Syntax

```
String hex(Bigint number)  
String hex(String number)  
String hex(BINARY number)
```

- Description

This function converts an integer or string into a hexadecimal number.

- Parameters

number: If number is of the BIGINT type, a hexadecimal number is returned. If number is of the STRING type, a string in hexadecimal format is returned.

- Return value

- If the input value is not 0 or NULL, a value of the STRING type is returned.
- If the input value is 0, 0 is returned.
- If the input value is NULL, an error is returned.

- Examples

```
hex(0)=0  
hex('abc')='616263'  
hex(17)='11'  
hex('17')='3137'  
hex(null): indicates that an exception occurs and the execution failed.
```

UNHEX

- Syntax

```
BINARY unhex(String number)
```

- **Description**

This function converts a hexadecimal string into a string.

- **Parameters**

number: the hexadecimal string.

- **Return value**

A value of the BINARY type is returned. If the input is 0, a failure is returned. If the input is NULL, NULL is returned.

- **Examples**

```
unhex('616263')='abc'
unhex(616263)='abc'
```

RADIANS

- **Syntax**

```
Double radians(Double number)
```

- **Description**

This function converts an angle to a radian value.

- **Parameters**

number: a value of the DOUBLE type.

- **Return value**

A value of the DOUBLE type is returned. If the input value is NULL, NULL is returned.

- **Examples**

```
radians(90)=1.5707963267948966
radians(0)=0.0
radians(null)=null
```

DEGREES

- **Syntax**

```
Double degrees(Double number)
Double degrees(Decimal number)
```

- **Description**

This function converts a radian value to an angle.

- **Parameters**

number: a value of the DOUBLE or DECIMAL type.

- **Return value**

A value of the DOUBLE type is returned. If the input value is NULL, NULL is returned.

- Examples

```
degrees(1.5707963267948966)=90.0
degrees(0)=0.0
degrees(null)=null
```

SIGN

- Syntax

```
Double sign(Double number)
Double sign(Decimal number)
```

- Description

This function obtains the sign of the input value.

- Parameters

number: a value of the DOUBLE or DECIMAL type.

- Return value

A value of the DOUBLE type is returned.

- If the input value is a positive value, 1.0 is returned.
- If the input value is a negative value, -1.0 is returned.
- If the input value is 0, 0.0 is returned.
- If the input value is NULL, NULL is returned.

- Examples

```
sign(-2.5)=-1.0
sign(2.5)=1.0
sign(0)=0.0
sign(null)=null
```

E

- Syntax

```
Double e()
```

- Description

This function calculates the value of `e`.

- Return value

A value of the DOUBLE type is returned.

- Examples

```
e()=2.718281828459045
```

PI

- Syntax

```
Double pi()
```

- Description

This function calculates the value of π .

- Return value

A value of the DOUBLE type is returned.

- Examples

```
pi()=3.141592653589793
```

FACTORIAL

- Syntax

```
Bigint factorial(Int number)
```

- Description

This function calculates the factorial of number.

- Parameters

number: a value of the INT type. The value ranges from 0 to 20.

- Return value

A value of the BIGINT type is returned. If the input value is 0, 1 is returned. If the input value is NULL or a value that does not fall into the range from 0 to 20, NULL is returned.

- Examples

```
factorial(5)=120 --5! =5*4*3*2*1=120
```

CBRT

- Syntax

```
Double cbrt(Double number)
```

- Description

This function calculates the cube root of number.

- Parameters

number: a value of the DOUBLE type.

- Return value

A value of the DOUBLE type is returned. If the input value is NULL, NULL is returned.

- Examples

```
cbirt(8)=2  
cbirt(null)=null
```

SHIFTLEFT

- Syntax

```
Int shiftleft(Tinyint|Smallint|Int number1, Int number2)  
Bigint shiftleft(Bigint number1, Int number2)
```

- Description

This function shifts a value left by a specific number of places (<<).

- Parameters

- number1: an integer of the TINYINT, SMALLINT, INT, or BIGINT type.
- number2: an integer of the INT type.

- Return value

A value of the INT or BIGINT type is returned.

- Examples

```
shiftleft(1,2)=4 -- Shifts the binary value of 1 two places to the left (1<<2, 0001 shifted to be 0100).  
shiftleft(4,3)=32 -- Shifts the binary value of 4 three places to the left (4<<3, 0100 shifted to be 100  
000).
```

SHIFTRIGHT

- Syntax

```
Int shiftright(Tinyint|Smallint|Int number1, Int number2)  
Bigint shiftright(Bigint number1, Int number2)
```

- Description

This function shifts a value right by a specific number of places (>>).

- Parameters

- number1: an integer of the TINYINT, SMALLINT, INT, or BIGINT type.
- number2: an integer of the INT type.

- Return value

A value of the INT or BIGINT type is returned.

- Examples

```
shiftright(4,2)=1 -- Shifts the binary value of 4 two places to the right (4>>2, 0100 shifted to be 0001  
).  
shiftright(32,3)=4 -- Shifts the binary value of 32 three places to the right (32>>3, 100000 shifted to  
be 0100).
```

SHIFTRIGHTUNSIGNED

- Syntax

```
Int shiftrightunsigned(Tinyint|Smallint|Int number1, Int number2)
Bigint shiftrightunsigned(Bigint number1, Int number2)
```

- Description

This function shifts an unsigned value right by a specific number of places (>>>).

- Parameters

- number1: an integer of the TINYINT, SMALLINT, INT, or BIGINT type.
- number2: an integer of the INT type.

- Return value

A value of the INT or BIGINT type is returned.

- Examples

```
shiftrightunsigned(8,2)=2 -- Shifts the binary unsigned value of 8 two places to the right (8>>>2, 10
00 shifted to be 0010).
shiftrightunsigned(-14,2)=1073741820 -- Shifts the binary value of -14 two places to the right (-14>>
>2, 11111111 11111111 11111111 11110010 shifted to be 00111111 11111111 11111111 11111100).
```

FORMAT_NUMBER

- Syntax

```
string format_number(float|double|decimal expr1, expr2)
```

- Description

This function converts a number to a string in a specified format.

- Parameters

- expr1: a numeric expression to format.
- expr2: the number of decimal places. It can be of the INT type. It can also be expressed in the format of `#,###,###.##`.
 - If expr2 is greater than 0, the value is rounded to the specified place after the decimal point.
 - If expr2 is equal to 0, the value has no decimal point or fractional part.
 - If expr2 is less than 0 or greater than 340, an error is returned.

- Return value

A value of the STRING type is returned.

- Examples

```
select format_number(5.230134523424545456,3);-- The return value is 5.230.
select format_number(12332.123456, '#,###,###,###.###');-- The return value is 12,332.123.
```

3.15.4. Window functions

You can use window functions to flexibly analyze and process jobs in MaxCompute SQL.

window functions



Notice

- Window functions can only be included in the `SELECT` clause.
- A window function cannot contain nested window or aggregate functions.
- Window functions cannot be used with aggregate functions of the same level.
- A maximum of five window functions can be used in a MaxCompute SQL statement.

Syntax of window functions

The syntax of a window function is as follows:

```
window_func() over (partition by [col1,col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] windowing_clause)
```

- `PARTITION BY` specifies partitioning columns.
- Rows with the same partitioning column value are considered in the same window. A window can contain a maximum of 100 million rows of data. However, we recommend that the number of rows in a window not exceed 5 million. If the number of rows exceeds the upper limit, an error is returned. This limit applies to all built-in window functions.



Note For example, `GROUP BY` `UID` indicates that records with the same `UID` are considered in the same window. Therefore, the records with the same `UID` can contain a maximum of 100 million rows of data. However, we recommend that the number of rows in a window not exceed 5 million.

- `ORDER BY` specifies the rule for sorting data in a window.



Note If some values in `ORDER BY` are the same, the sorting result is unstable. To reduce randomness, keep each value in `ORDER BY` unique.

- You can use `ROWS` in `windowing_clause` to specify the partitioning method. There are two methods:
 - `rows between x preceding|following and y preceding|following` : indicates a window range from the x^{th} row preceding or following the current row to the y^{th}
 - `rows x preceding|following` : indicates a window range from the x^{th} row preceding or following the current row to the current row.

 Note

- `x` and `y` must be integer constants greater than or equal to 0. Their values range from 0 to 10000. 0 indicates the current row. You must specify `ORDER BY` before you use `ROWS` to specify a window range.
- Only the following window functions can use `ROWS` to specify the partitioning method: AVG, COUNT, MAX, MIN, STDDEV, and SUM.

COUNT

The syntax, description, and example of the COUNT function are as follows:

- **Syntax**

```
Bigint count([distinct] expr) over(partition by [col1, col2...]  
[order by [col1[asc|desc], col2[asc|desc]...] [windowing_clause])
```

- **Description**

Calculates the number of rows that meet `expr`.

- **Parameters**

- `expr` : a value of any data type. If the value for a row is NULL, the row is not included in the calculation. If the `distinct` keyword is specified, this parameter indicates that only distinct values are counted.
- `partition by [col1, col2...]` : specifies partitioning columns.
- `order by col1 [asc|desc], col2[asc|desc]` : If `ORDER BY` is not specified, the number of rows that meet `expr` in the current window is returned. If `ORDER BY` is specified, the returned results are sorted in the specified order and the number of rows from the starting row to the current row in the current window is returned.

 **Note** `ORDER BY` cannot be used if the `distinct` keyword is specified.

- **Return value**

Returns a value of the BIGINT type.

- **Example**

Assume that the `test_src` table contains the `user_id` column of the BIGINT type.

```
select user_id,count(user_id) over (partition by user_id) as count
from test_src;
```

```
+-----+-----+
```

```
| user_id | count |
```

```
+-----+-----+
```

```
| 1 | 3 |
```

```
| 1 | 3 |
```

```
| 1 | 3 |
```

```
| 2 | 1 |
```

```
| 3 | 1 |
```

```
+-----+-----+
```

-- If ORDER BY is not specified, the number of rows of the user_id column from the current window is returned.

```
select user_id,count(user_id) over (partition by user_id order by user_id) as count
from test_src;
```

```
+-----+-----+
```

```
| user_id | count |
```

```
+-----+-----+
```

```
| 1 | 1 | -- This row is the starting row of this window.
```

```
| 1 | 2 | -- Two records exist from the starting row to the current row. The number 2 is returned.
```

```
| 1 | 3 |
```

```
| 2 | 1 |
```

```
| 3 | 1 |
```

```
+-----+-----+
```

-- If ORDER BY is specified, the number of rows from the starting row to the current row in the current window is returned.

If duplicate values are specified for `ORDER BY`, the processing method varies depending on whether MaxCompute is compatible with Hive.

- If MaxCompute is incompatible with Hive, the return value is the value of COUNT for each row.

```
set odps.sql.hive.compatible=false;
select user_id, price, count(price) over
(partition by user_id order by price) as count from test_src;
```

```
+-----+-----+-----+
| user_id | price | count |
+-----+-----+-----+
| 1       | 4.5   | 1     | -- This row is the starting row of this window.
| 1       | 5.5   | 2     | -- The value of COUNT for the second row is 2.
| 1       | 5.5   | 3     | -- The value of COUNT for the third row is 3.
| 1       | 6.5   | 4     |
| 2       | NULL  | 0     |
| 2       | 3.0   | 1     |
| 3       | NULL  | 0     |
| 3       | 4.0   | 1     |
+-----+-----+-----+
```

- If MaxCompute is compatible with Hive, the return value is the value of COUNT for the last row of the rows with same values.

```
set odps.sql.hive.compatible=true;
select user_id, price, count(price) over
(partition by user_id order by price) as count from test_src;
```

```
+-----+-----+-----+
| user_id | price | count |
+-----+-----+-----+
| 1       | 4.5   | 1     | -- This row is the starting row of this window.
| 1       | 5.5   | 3     | -- The value of COUNT for the second row is the value of COUNT for the thi
rd row because the prices of the two rows are the same.
| 1       | 5.5   | 3     | -- The value of COUNT for the third row is 3.
| 1       | 6.5   | 4     |
| 2       | NULL  | 0     |
| 2       | 3.0   | 1     |
| 3       | NULL  | 0     |
| 3       | 4.0   | 1     |
+-----+-----+-----+
```

AVG

The syntax, description, and example of the AVG function are as follows:

- Syntax


```
avg([distinct] expr) over(partition by [col1, col2...]  
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
```

- Description

Calculates the average value of input values.

- Parameters

- `distinct` : If the `distinct` keyword is specified, this parameter indicates that the average value of distinct values is calculated.
- `expr` : a value of the DOUBLE or DECIMAL type.
 - If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If it is of another data type, an error is returned.
 - If the value for a row is NULL, the row is not included in the calculation.
 - The values of the BOOLEAN type are not included in the calculation.
- `partition by [col1, col2...]` specifies partitioning columns.
- `order by col1[asc|desc], col2[asc|desc]` : If `ORDER BY` is not specified, the average value of all values in the current window is returned. If `ORDER BY` is specified, the returned results are sorted in the specified order and the average value from the starting row to the current row in the current window is returned.

 **Note** `ORDER BY` cannot be used if the `distinct` keyword is specified.

- Return value

Returns a value of the DOUBLE type.

If duplicate values are specified for `ORDER BY`, the processing method varies depending on whether MaxCompute is compatible with Hive.

- If MaxCompute is incompatible with Hive, the return value is the average value for each row.

```

set odps.sql.hive.compatible=false;
select user_id, price, avg(price) over
(partition by user_id order by price) from test_src;

+-----+-----+-----+
| user_id | price | _c2      |
+-----+-----+-----+
| 1       | 4.5   | 4.5       | -- This row is the starting row of this window.
| 1       | 5.5   | 5.0       | -- The return value is the average value of the values for the first an
d second rows.
| 1       | 5.5   | 5.166666666666667 | -- The return value is the average value of the values from t
he first row to the third row.
| 1       | 6.5   | 5.5       |
| 2       | NULL  | NULL      |
| 2       | 3.0   | 3.0       |
| 3       | NULL  | NULL      |
| 3       | 4.0   | 4.0       |
+-----+-----+-----+

```

- If MaxCompute is compatible with Hive, the return value is the average value of values for the last row of the rows with same values.

```

set odps.sql.hive.compatible=true;
select user_id, price, avg(price) over
(partition by user_id order by price) from test_src;

+-----+-----+-----+
| user_id | price | _c2      |
+-----+-----+-----+
| 1       | 4.5   | 4.5       | -- This row is the starting row of this window.
| 1       | 5.5   | 5.166666666666667 | -- The return value is the average value of values from the fir
st row to the third row.
| 1       | 5.5   | 5.166666666666667 | -- The return value is the average value of values from the fir
st row to the third row.
| 1       | 6.5   | 5.5       |
| 2       | NULL  | NULL      |
| 2       | 3.0   | 3.0       |
| 3       | NULL  | NULL      |
| 3       | 4.0   | 4.0       |
+-----+-----+-----+

```

MAX

- Syntax

```
max([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
```

- Description

Calculates the maximum value of input values.

- Parameters

- `expr` : a value of any data type except BOOLEAN. If the value for a row is NULL, the row is not included in the calculation. If the `distinct` keyword is specified, this parameter indicates that the maximum value of distinct values is calculated. The calculation result is not affected by whether the parameter is set.
- `partition by [col1, col2...]` : specifies partitioning columns.
- `order by [col1[asc|desc], col2[asc|desc] : If ORDER BY is not specified, the maximum value of all values in the current window is returned. If ORDER BY is specified, the returned results are sorted in the specified order and the maximum value of values from the starting row to the current row in the current window is returned.`

 **Note** `ORDER BY` cannot be used if the `distinct` keyword is specified.

- Return value

Returns values of the same data type as `expr` .

MIN

- Syntax

```
min([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
```

- Description

Calculates the minimum value of input values.

- Parameters

- `expr` : a value of any data type except BOOLEAN. If the value for a row is NULL, the row is not included in the calculation. If the `distinct` keyword is specified, this parameter indicates that the minimum value of distinct values is calculated. The calculation result is not affected by whether the parameter is set.
- `partition by [col1, col2...]` : specifies partitioning columns.
- `order by [col1[asc|desc], col2[asc|desc] : If ORDER BY is not specified, the minimum value in the current window is returned. If ORDER BY is specified, the returned results are sorted in the specified order and the minimum value of values from the starting row to the current row in the current window is returned.`

 **Note** `ORDER BY` cannot be used if the `distinct` keyword is specified.

- Return value

Returns values of the same data type as `expr`.

MEDIAN

- Syntax

```
Double median(Double number1,number2...) over(partition by [col1, col2...])
Decimal median(Decimal number1,number2...) over(partition by [col1,col2...])
```

- Description

Calculates the minimum value of median values.

- Parameters

- `number1,number1...:` numbers of the DOUBLE or DECIMAL type. You can enter 1 to 255 numbers.
 - If the input value is of the DOUBLE type, it is converted into an array of the DOUBLE type for calculation by default.
 - If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If it is of another data type, an error is returned.
 - If the input value is NULL, NULL is returned.
- `partition by [col1, col2...]` : specifies partitioning columns.

- Return value

Returns a value of the DOUBLE or DECIMAL type.

STDDEV

The syntax, description, and example of the STDDEV function are as follows:

- Syntax

```
Double stddev([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
Decimal stddev([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
```

- Description

Calculates the population standard deviation.

- Parameters

- `expr` : a value of the DOUBLE or DECIMAL type.
 - If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If it is of another data type, an error is returned.
 - If the value for a row is NULL, the row is not included in the calculation.
 - If the `distinct` keyword is specified, this parameter indicates that the population standard deviation of distinct values is calculated.

 Note

- `ORDER BY` cannot be used if the `distinct` keyword is specified.
- `stddev` has an alias function `stddev_pop`, which is used the same as `stddev`.

- `partition by [col1, col2..]` : specifies partitioning columns.
- `order by col1[asc|desc], col2[asc|desc]` : If `ORDER BY` is not specified, the population standard deviation of the current window is returned. If `ORDER BY` is specified, the returned results are sorted in the specified order and the population standard deviation from the starting row to the current row in the current window is returned.

- **Return value**

If the input value is of the DECIMAL type, a value of the DECIMAL type is returned. Otherwise, a value of the DOUBLE type is returned.

- **Example**

```
select window, seq, stddev_pop('1\01') over (partition by window order by seq) from dual;
```

If duplicate values are specified for `ORDER BY`, the processing method varies depending on whether MaxCompute is compatible with Hive.

- If MaxCompute is incompatible with Hive, the return value is the value of STDDEV for each row.

```

set odps.sql.hive.compatible=false;
select user_id, price, stddev(price) over
(partition by user_id order by price) from test_src;

+-----+-----+-----+
| user_id | price | _c2      |
+-----+-----+-----+
| 1       | 4.5   | 0.0      | -- This row is the starting row of this window.
| 1       | 5.5   | 0.5      | -- The return value is the value of STDDEV for the first and second rows.
| 1       | 5.5   | 0.4714045207910316 | -- The return value is the value of STDDEV from the first row to the third row.
| 1       | 6.5   | 0.7071067811865475 |
| 2       | NULL  | NULL     |
| 2       | 3.0   | 0.0      |
| 3       | NULL  | NULL     |
| 3       | 4.0   | 0.0      |
+-----+-----+-----+

```

- If MaxCompute is compatible with Hive, the return value is the value of STDDEV for the last row of the rows with same values.

```

set odps.sql.hive.compatible=true;
select user_id, price, stddev(price) over
(partition by user_id order by price) from test_src;

+-----+-----+-----+
| user_id | price | _c2      |
+-----+-----+-----+
| 1       | 4.5   | 0.0      | -- This row is the starting row of this window.
| 1       | 5.5   | 0.4714045207910316 | -- The return value is the value of STDDEV from the first row to the third row.
| 1       | 5.5   | 0.4714045207910316 | -- The return value is the value of STDDEV from the first row to the third row.
| 1       | 6.5   | 0.7071067811865475 |
| 2       | NULL  | NULL     |
| 2       | 3.0   | 0.0      |
| 3       | NULL  | NULL     |
| 3       | 4.0   | 0.0      |
+-----+-----+-----+

```

STDDEV_SAMP

- Syntax

```
Double stddev_samp([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
Decimal stddev_samp([distinct] expr) over((partition by [col1,col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
```

- Description

Calculates the sample standard deviation.

- Parameters

- `expr` : a value of the DOUBLE or DECIMAL type.
 - If the input value is of the STRING or BIGINT type, it is implicitly converted into the DOUBLE type before calculation. If it is of another data type, an error is returned.
 - If the value for a row is NULL, the row is not included in the calculation.
 - If the `distinct` keyword is specified, this parameter indicates that the sample standard deviation of distinct values is calculated.

 **Note** `ORDER BY` cannot be used if the `distinct` keyword is specified.

- `partition by [col1, col2..]` : specifies partitioning columns.
- `order by col1[asc|desc], col2[asc|desc]` : If `ORDER BY` is not specified, the sample standard deviation in the current window is returned. If `ORDER BY` is specified, the returned results are sorted in the specified order and the sample standard deviation from the starting row to the current row in the current window is returned.

- Return value

If the input value is of the DECIMAL type, a value of the DECIMAL type is returned. Otherwise, a value of the DOUBLE type is returned.

SUM

The syntax, description, and example of the SUM function are as follows:

- Syntax

```
sum([distinct] expr) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]] [windowing_clause])
```

- Description

Calculates the sum of input values.

- Parameters

- `expr` : a value of the DOUBLE, DECIMAL, or BIGINT type.
 - If the input value is of the STRING type, it is implicitly converted into the DOUBLE type before calculation. If it is of another data type, an error is returned.
 - If the value for a row is NULL, the row is not included in the calculation.
 - If the `distinct` keyword is specified, this parameter indicates that the sum of distinct values is calculated.
- `partition by [col1, col2..]` : specifies partitioning columns.
- `order by col1[asc|desc], col2[asc|desc]` : If `ORDER BY` is not specified, the sum of the `expr` value in the current window is returned. If `ORDER BY` is specified, the returned results are sorted in the specified order and the sum of values from the first row to the current row in the current window is returned.

 **Note** `ORDER BY` cannot be used if the `distinct` keyword is specified.

- **Return value**
 - If the input value is of the BIGINT type, a value of the BIGINT type is returned.
 - If the input value is of the DECIMAL type, a value of the DECIMAL type is returned.
 - If the input value is of the DOUBLE or STRING type, a value of the DOUBLE type is returned.

If duplicate values are specified for `ORDER BY`, the processing method varies depending on whether MaxCompute is compatible with Hive.

- If MaxCompute is incompatible with Hive, the return values are different. The return value is the sum of values for each row.

```
set odps.sql.hive.compatible=false;
select user_id, price, sum(price) over
(partition by user_id order by price) from test_src;
```

```
+-----+-----+-----+
| user_id | price | _c2 |
+-----+-----+-----+
| 1 | 4.5 | 4.5 | -- This row is the starting row of this window.
| 1 | 5.5 | 10.0 | -- The return value is the sum of values for the first and second rows.
| 1 | 5.5 | 15.5 | -- The return value is the sum of values from the first row to the third row.
| 1 | 6.5 | 22.0 |
| 2 | NULL | NULL |
| 2 | 3.0 | 3.0 |
| 3 | NULL | NULL |
| 3 | 4.0 | 4.0 |
+-----+-----+-----+
```


- If MaxCompute is compatible with Hive, the return values are the same. The return value is the sum of values for the last row of the rows with same values.

```

set odps.sql.hive.compatible=true;
select user_id, price, sum(price) over
  (partition by user_id order by price) from test_src;
+-----+-----+-----+
| user_id | price | _c2 |
+-----+-----+-----+
| 1       | 4.5   | 4.5   | -- This row is the starting row of this window.
| 1       | 5.5   | 15.5  | -- The return value is the sum of values from the first row to the third row.
| 1       | 5.5   | 15.5  | -- The return value is the sum of values from the first row to the third row.
| 1       | 6.5   | 22.0   |
| 2       | NULL  | NULL   |
| 2       | 3.0   | 3.0    |
| 3       | NULL  | NULL   |
| 3       | 4.0   | 4.0    |
+-----+-----+-----+

```

DENSE_RANK

- Syntax

```

Bigint dense_rank() over(partition by [col1, col2...]
order by [col1[asc|desc], col2[asc|desc]...])

```

- Description

Calculates continuous rankings. The rankings for the data in rows with the same `col2` are the same.

- Parameters

- `partition by [col1, col2..]` : specifies partitioning columns.
- `order by col1[asc|desc], col2[asc|desc]` : specifies the field based on which data is ranked.

- Return value

Returns a value of the `BIGINT` type.

- Example

Assume that table `emp` has the following data:

```
| empno | ename | job | mgr | hiredate| sal| comm | deptno |
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
7948,JACCKA,CLERK,7782,1981-04-12 00:00:00,5000,,10
7956,WELAN,CLERK,7649,1982-07-20 00:00:00,2450,,10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10
```

Group employees by department, sort the employees in each group in descending order of `sal`, and then obtain the sequence numbers of employees in each group.

-- deptno that specifies a department is used as a partitioning column. sal that specifies the salary is used as the value to be sorted in the returned result.

```
SELECT deptno, ename, sal, DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS nums
FROM emp;
```

-- The result is as follows:

```
+-----+-----+-----+-----+
| deptno | ename | sal   | nums |
+-----+-----+-----+-----+
| 10     | JACCKA | 5000.0 | 1    |
| 10     | KING  | 5000.0 | 1    |
| 10     | CLARK | 2450.0 | 2    |
| 10     | WELAN | 2450.0 | 2    |
| 10     | TEBAGE | 1300.0 | 3    |
| 10     | MILLER | 1300.0 | 3    |
| 20     | SCOTT | 3000.0 | 1    |
| 20     | FORD  | 3000.0 | 1    |
| 20     | JONES | 2975.0 | 2    |
| 20     | ADAMS | 1100.0 | 3    |
| 20     | SMITH | 800.0  | 4    |
| 30     | BLAKE | 2850.0 | 1    |
| 30     | ALLEN | 1600.0 | 2    |
| 30     | TURNER | 1500.0 | 3    |
| 30     | MARTIN | 1250.0 | 4    |
| 30     | WARD  | 1250.0 | 4    |
| 30     | JAMES | 950.0  | 5    |
+-----+-----+-----+-----+
```

RANK

- Syntax

```
Bigint rank() over(partition by [col1, col2...]
order by [col1[asc|desc], col2[asc|desc]...])
```

- Description

Returns the rankings of data in a window. If two rows have the same col2, the ranking of the second row drops.

- Parameters

- partition by [col1, col2.] : specifies partitioning columns.
- order by col1[asc|desc], col2[asc|desc] : specifies the field based on which data is ranked.

- Return value

Returns a value of the BIGINT type.

- Example

Assume that table `emp` has the following data:

```
| empno | ename | job | mgr | hiredate| sal| comm | deptno |
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
7948,JACCKA,CLERK,7782,1981-04-12 00:00:00,5000,,10
7956,WELAN,CLERK,7649,1982-07-20 00:00:00,2450,,10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10
```

Group employees by department, sort the employees in each group in descending order of `sal`, and then obtain the sequence numbers of employees in each group.

-- deptno that specifies a department is used as a partitioning column. sal that specifies the salary is used as the value to be sorted in the returned result.

```
SELECT deptno,ename,sal, RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS nums
FROM emp;
```

-- The result is as follows:

deptno	ename	sal	nums
10	JACCKA	5000.0	1
10	KING	5000.0	1
10	CLARK	2450.0	3
10	WELAN	2450.0	3
10	TEBAGE	1300.0	5
10	MILLER	1300.0	5
20	SCOTT	3000.0	1
20	FORD	3000.0	1
20	JONES	2975.0	3
20	ADAMS	1100.0	4
20	SMITH	800.0	5
30	BLAKE	2850.0	1
30	ALLEN	1600.0	2
30	TURNER	1500.0	3
30	MARTIN	1250.0	4
30	WARD	1250.0	4
30	JAMES	950.0	6

LAG

- Syntax

```
lag(expr, Bigint offset, default) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]])
```

- Description

Returns the value for a row at a given offset preceding the current row. If the current row number is `rn`, the value of the row with the number of `rn-offset` is retrieved.

- Parameters

- `expr`: a value of any data type.
- `offset`: a constant of the BIGINT type. The value of the offset must be greater than 0. If the input value is of the STRING or DOUBLE type, it is implicitly converted into the BIGINT type before calculation.

- **default**: the default value that is used if offset is out of the valid range. It is a constant value and its default value is NULL.
- **partition by [col1, col2..]** : specifies partitioning columns.
- **order by col1[asc|desc], col2[asc|desc]** : specifies the field based on which data is ranked.
- **Return value**
Returns a value of the same data type as `expr` .
- **Example**

```
select seq, lag(seq+100, 1) over (partition by window order by seq) as r from sliding_window;
```

-- The result is as follows:

```
+-----+-----+
| seq  | r    |
+-----+-----+
| 0    | NULL |
| 1    | 100  |
| 2    | 101  |
| 3    | 102  |
| 4    | 103  |
| 5    | 104  |
| 6    | 105  |
| 7    | 106  |
| 8    | 107  |
| 9    | 108  |
+-----+-----+
```

LEAD

- **Syntax**

```
lead(expr, Bigint offset, default) over(partition by [col1, col2...]
[order by [col1[asc|desc], col2[asc|desc]...]])
```

- **Description**

Retrieves the value for a row at a given offset following the current row. If the current row number is `rn` , the value of the row with the number of `rn+offset` is retrieved.

- **Parameters**

- **expr** : a value of any data type.
- **offset**: a constant of the BIGINT type. The value of the offset must be greater than 0. If the input value is of the STRING or DOUBLE type, it is implicitly converted into the BIGINT type before calculation.
- **default**: the default value that is used if offset is out of the valid range. It is a constant value and its default value is NULL.

- `partition by [col1, col2..]` : specifies partitioning columns.
- `order by col1[asc|desc], col2[asc|desc]` : specifies the field based on which data is ranked.

- Return value

Returns a value of the same data type as `expr` .

- Example

```
select c_Double_a,c_String_b,c_int_a,lead(c_int_a,1) over(partition by c_Double_a order by c_String_b
) from dual;
select c_String_a,c_time_b,c_Double_a,lead(c_Double_a,1) over(partition by c_String_a order by c_time_b) from dual;
select c_String_in_fact_num,c_String_a,c_int_a,lead(c_int_a) over(partition by c_String_in_fact_num order by c_String_a) from dual;
```

PERCENT_RANK

- Syntax

```
percent_rank() over(partition by [col1, col2...]
order by [col1[asc|desc], col2[asc|desc]...])
```

- Description

Calculates the relative ranking of a row in a group of data.

- Parameters

- `partition by [col1, col2..]` : specifies partitioning columns.
- `order by col1[asc|desc], col2[asc|desc]` : specifies the field based on which data is ranked.

- Return value

Returns a value of the DOUBLE type. Value range: [0, 1]. The relative ranking is calculated by using the following formula: $(\text{Rank} - 1) / (\text{Number of rows} - 1)$.

ROW_NUMBER

- Syntax

```
row_number() over(partition by [col1, col2...]
order by [col1[asc|desc], col2[asc|desc]...])
```

- Description

Calculates the row number, starting from 1.

- Parameters

- `partition by [col1, col2..]` : specifies partitioning columns.
- `order by col1[asc|desc], col2[asc|desc]` : specifies the field based on which data is ranked.

- Return value

Returns a value of the BIGINT type.

- Example

Assume that table `emp` has the following data:

```
| empno | ename | job | mgr | hiredate| sal| comm | deptno |
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
7948,JACCKA,CLERK,7782,1981-04-12 00:00:00,5000,,10
7956,WELAN,CLERK,7649,1982-07-20 00:00:00,2450,,10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10
```

Group employees by department, sort the employees in each group in descending order of `sal`, and then obtain the sequence numbers of employees in each group.


```
SELECT deptno,ename,sal,ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sal DESC) AS nums
-- deptno that specifies a department is used as a partitioning column. sal that specifies the salary i
s used as the value to be sorted in the returned result.
```

```
FROM emp;
```

```
-- The result is as follows:
```

```
+-----+-----+-----+-----+
| deptno | ename | sal   | nums |
+-----+-----+-----+-----+
| 10     | JACCKA | 5000.0 | 1     |
| 10     | KING  | 5000.0 | 2     |
| 10     | CLARK  | 2450.0 | 3     |
| 10     | WELAN  | 2450.0 | 4     |
| 10     | TEBAGE | 1300.0 | 5     |
| 10     | MILLER | 1300.0 | 6     |
| 20     | SCOTT  | 3000.0 | 1     |
| 20     | FORD   | 3000.0 | 2     |
| 20     | JONES  | 2975.0 | 3     |
| 20     | ADAMS  | 1100.0 | 4     |
| 20     | SMITH  | 800.0  | 5     |
| 30     | BLAKE  | 2850.0 | 1     |
| 30     | ALLEN  | 1600.0 | 2     |
| 30     | TURNER | 1500.0 | 3     |
| 30     | MARTIN | 1250.0 | 4     |
| 30     | WARD   | 1250.0 | 5     |
| 30     | JAMES  | 950.0  | 6     |
+-----+-----+-----+-----+
```

CLUSTER_SAMPLE

- Syntax

```
boolean cluster_sample([Bigint x, Bigint y])
over(partition by [col1, col2..])
```

- Description

Performs group sampling on data in a window.

- Parameters

- **x**: a constant of the BIGINT type. The value of **x** must be greater than or equal to 1. If **y** is specified, **x** indicates that a window is divided into **x** portions. Otherwise, **x** indicates that the records of **x** rows in a window are extracted. In this case, True is returned for the **x** rows. If **x** is NULL, NULL is returned.

- **y**: a constant of the BIGINT type. The value of **y** must be greater than or equal to 1 and must be less than or equal to **x**. **y** indicates that the extracted **y** records of the **x** portions in a window are extracted. In this case, True is returned for the **y** records. If **y** is set to NULL, NULL is returned.
- **partition by [col1, col2]** : specifies partitioning columns.
- **Return value**
Returns a value of the BOOLEAN type.
- **Example**

For example, if table **test_tbl** has the **key** and **value** columns, **key** is the grouping field and the value options include **groupa** and **groupb**. **value** is the key value, as shown in the following code:

```
+-----+-----+
| key   | value   |
+-----+-----+
| groupa | -1.34764165478145 |
| groupa | 0.740212609046718 |
| groupa | 0.167537127858695 |
| groupa | 0.630314566185241 |
| groupa | 0.0112401388646925 |
| groupa | 0.199165745875297 |
| groupa | -0.320543343353587 |
| groupa | -0.273930924365012 |
| groupa | 0.386177958942063 |
| groupa | -1.09209976687047 |
| groupb | -1.10847690938643 |
| groupb | -0.725703978381499 |
| groupb | 1.05064697475759 |
| groupb | 0.135751224393789 |
| groupb | 2.13313102040396 |
| groupb | -1.11828960785008 |
| groupb | -0.849235511508911 |
| groupb | 1.27913806620453 |
| groupb | -0.330817716670401 |
| groupb | -0.300156896191195 |
| groupb | 2.4704244205196 |
| groupb | -1.28051882084434 |
+-----+-----+
```

If you want to extract a sample of 10% of the values in each group, execute the following MaxCompute SQL statement:

```

select key, value
  from (
    select key, value, cluster_sample(10, 1) over(partition by key) as flag
    from tbl
  ) sub
 where flag = true;
-- The result is as follows:
+-----+-----+
| key | value |
+-----+-----+
| groupa | 0.167537127858695 |
| groupb | 0.135751224393789 |
+-----+-----+

```

CUME_DIST

- Syntax

```
cume_dist() over(partition by col1[, col2...] order by col1 [asc|desc][, col2[asc|desc]...])
```

- Description

Calculates the cumulative distribution. The cumulative distribution is the ratio of rows whose values are greater than or equal to the current value to all rows in a group.

- Parameters

The **ORDER BY** column specifies the values to be compared.

- Return value

Returns the ratio of rows whose values are greater than or equal to the current value to all rows in a group.

- Example

Assume that table **emp** has the following data:

```
| empno | ename | job | mgr | hiredate| sal| comm | deptno |
7369,SMITH,CLERK,7902,1980-12-17 00:00:00,800,,20
7499,ALLEN,SALESMAN,7698,1981-02-20 00:00:00,1600,300,30
7521,WARD,SALESMAN,7698,1981-02-22 00:00:00,1250,500,30
7566,JONES,MANAGER,7839,1981-04-02 00:00:00,2975,,20
7654,MARTIN,SALESMAN,7698,1981-09-28 00:00:00,1250,1400,30
7698,BLAKE,MANAGER,7839,1981-05-01 00:00:00,2850,,30
7782,CLARK,MANAGER,7839,1981-06-09 00:00:00,2450,,10
7788,SCOTT,ANALYST,7566,1987-04-19 00:00:00,3000,,20
7839,KING,PRESIDENT,,1981-11-17 00:00:00,5000,,10
7844,TURNER,SALESMAN,7698,1981-09-08 00:00:00,1500,0,30
7876,ADAMS,CLERK,7788,1987-05-23 00:00:00,1100,,20
7900,JAMES,CLERK,7698,1981-12-03 00:00:00,950,,30
7902,FORD,ANALYST,7566,1981-12-03 00:00:00,3000,,20
7934,MILLER,CLERK,7782,1982-01-23 00:00:00,1300,,10
7948,JACCKA,CLERK,7782,1981-04-12 00:00:00,5000,,10
7956,WELAN,CLERK,7649,1982-07-20 00:00:00,2450,,10
7956,TEBAGE,CLERK,7748,1982-12-30 00:00:00,1300,,10
```

Group all employees by department and obtain the cumulative distribution of `sal` for each group.

```
SELECT deptno
,   ename
,   sal
,   concat(round(cume_dist() OVER(PARTITION BY deptno ORDER BY sal desc)*100,2),'%') as cume_dist
FROM emp;
```

-- The result is as follows:

deptno	ename	sal	cume_dist
10	JACCKA	5000.0	33.33%
10	KING	5000.0	33.33%
10	CLARK	2450.0	66.67%
10	WELAN	2450.0	66.67%
10	TEBAGE	1300.0	100.0%
10	MILLER	1300.0	100.0%
20	SCOTT	3000.0	40.0%
20	FORD	3000.0	40.0%
20	JONES	2975.0	60.0%
20	ADAMS	1100.0	80.0%
20	SMITH	800.0	100.0%
30	BLAKE	2850.0	16.67%
30	ALLEN	1600.0	33.33%
30	TURNER	1500.0	50.0%
30	MARTIN	1250.0	83.33%
30	WARD	1250.0	83.33%
30	JAMES	950.0	100.0%

FIRST_VALUE

The syntax, description, and example of the FIRST_VALUE function are as follows:

- Syntax

```
first_value(expr) over(partition by col1[, col2...]
[order by col1 [asc|desc][, col2[asc|desc]...]] [windowing_clause])
```

- Description

Calculates the first value of input values.

- Parameters

- expr: a value of any basic data type.
- partition by col1[, col2...]: specifies partitioning columns.

- `order by col1[asc|desc], col2[asc|desc]`: If `ORDER BY` is not specified, the value of `expr` of the starting row in the current window is returned. If `ORDER BY` is specified, the returned results are sorted in the specified order and the value of `expr` of the starting row in the current window is returned.
- Return value
Returns a value of the same data type as `expr`.
- Example

```
select user_id, price, first_value(price) over
(partition by user_id) as first_value from test_src;
```

```
+-----+-----+-----+
| user_id | price | first_value |
+-----+-----+-----+
| 1      | 5.5   | 5.5         | -- This row is the starting row of this window.
| 1      | 4.5   | 5.5         |
| 1      | 6.5   | 5.5         |
| 1      | 5.5   | 5.5         |
| 2      | NULL  | NULL        | -- This row is the starting row of this window.
| 2      | 3.0   | NULL        |
| 3      | 4.0   | 4.0         | -- This row is the starting row of this window.
| 3      | NULL  | 4.0         |
```

-- If ORDER BY is not specified, rows from the first row to the last row belong to the current window.
The value of the starting row in the current window is returned.

```
select user_id, price, first_value(price) over
(partition by user_id order by price) as first_value from test_src;
```

```
+-----+-----+-----+
| user_id | price | first_value |
+-----+-----+-----+
| 1      | 4.5   | 4.5         | -- This row is the starting row of this window.
| 1      | 5.5   | 4.5         |
| 1      | 5.5   | 4.5         |
| 1      | 6.5   | 4.5         |
| 2      | NULL  | NULL        | -- This row is the starting row of this window.
| 2      | 3.0   | NULL        |
| 3      | NULL  | NULL        | -- This row is the starting row of this window.
| 3      | 4.0   | NULL        |
```

-- If ORDER BY is specified, rows from the first row to the current row belong to the current window.
The value of the starting row in the current window is returned.

LAST_VALUE

The syntax, description, and example of the LAST_VALUE function are as follows:

- Syntax

```
last_value(expr) over(partition by col1[, col2...]  
    [order by col1 [asc|desc][, col2[asc|desc]...]] [windowing_clause])
```

- Syntax

Calculates the last value of input values.

- Parameters

- expr: a value of any basic data type.
- partition by col1[, col2...]: specifies partitioning columns.
- order by col1[asc|desc], col2[asc|desc]: If **ORDER BY** is not specified, the value of expr of the last row in the current window is returned. If **ORDER BY** is specified, the returned results are sorted in the specified order and the value of expr of the current row in the current window is returned.

- Return value

Returns a value of the same data type as expr.

- Example


```
select user_id, price, last_value(price) over
(partition by user_id) as last_value from test_src;
```

```
+-----+-----+-----+
| user_id | price | last_value |
+-----+-----+-----+
| 1       | 5.5   | 5.5       |
| 1       | 4.5   | 5.5       |
| 1       | 6.5   | 5.5       |
| 1       | 5.5   | 5.5       | -- This row is the last row in this window.
| 2       | NULL  | 3.0       |
| 2       | 3.0   | 3.0       | -- This row is the last row in this window.
| 3       | 4.0   | NULL      |
| 3       | NULL  | NULL      | -- This row is the last row in this window.
```

```
+-----+-----+-----+
-- If ORDER BY is not specified, the rows from the first row to the last row belong to the current window, and the value of the last row in the current window is returned.
```

```
select user_id, price, last_value(price) over
(partition by user_id order by price) as last_value from test_src;
```

```
+-----+-----+-----+
| user_id | price | last_value |
+-----+-----+-----+
| 1       | 4.5   | 4.5       | -- This row is the current row in the current window.
| 1       | 5.5   | 5.5       | -- This row is the current row in the current window.
| 1       | 5.5   | 5.5       | -- This row is the current row in the current window.
| 1       | 6.5   | 6.5       | -- This row is the current row in the current window.
| 2       | NULL  | NULL      | -- This row is the current row in the current window.
| 2       | 3.0   | 3.0       | -- This row is the current row in the current window.
| 3       | NULL  | NULL      | -- This row is the current row in the current window.
| 3       | 4.0   | 4.0       | -- This row is the current row in the current window.
```

```
+-----+-----+-----+
-- If ORDER BY is specified, rows from the first row to the current row belong to the current window. The value of the current row in the current window is returned.
```

NTH_VALUE

The syntax, description, and example of the NTH_VALUE function are as follows:

- Syntax

```
nth_value(expr, number [, skipNull]) over(partition by col1[, col2...]
[order by col1 [asc|desc][, col2[asc|desc]...] [windowing_clause])
```

- Description

Calculates the n^{th} value of input values. If the value of n exceeds the total number of rows in the window, NULL is returned.

- Parameters

- **expr**: a value of any basic data type.
- **number**: an integer greater than or equal to 1.
- **skipNull**: indicates whether to skip NULL when the n^{th} value is calculated. The default value is *false*.
- **partition by col1[, col2...]**: specifies partitioning columns.
- **order by col1[asc|desc], col2[asc|desc]**: If **ORDER BY** is not specified, the value of **expr** of the n^{th} row in the current window is returned. If **ORDER BY** is specified, the returned results are sorted in the specified order and the value of **expr** of the n^{th} row from the starting row to the current row in the current window is returned.

- Return value

Returns a value of the same data type as **expr**.

- Example

```
select user_id, price, nth_value(price, 2) over
(partition by user_id) as nth_value from test_src;
```

```
+-----+-----+-----+
| user_id | price | nth_value |
+-----+-----+-----+
| 1       | 5.5   | 4.5       |
| 1       | 4.5   | 4.5       | -- This row is the second row in the current window.
| 1       | 6.5   | 4.5       |
| 1       | 5.5   | 4.5       |
| 2       | NULL  | 3.0       |
| 2       | 3.0   | 3.0       | -- This row is the second row in the current window.
| 3       | 4.0   | NULL      |
| 3       | NULL  | NULL      | -- This row is the second row in the current window.
```

```
+-----+-----+-----+
-- If ORDER BY is not specified, rows from the first row to the last row belong to the current window.
The value of the second row is returned.
```

```
select user_id, price, nth_value(price, 3) over
(partition by user_id) as nth_value from test_src;
```

```
+-----+-----+-----+
```

```

| user_id | price | nth_value |
+-----+-----+-----+
| 1      | 5.5   | 6.5       |
| 1      | 4.5   | 6.5       |
| 1      | 6.5   | 6.5       | -- This row is the third row in the current window.
| 1      | 5.5   | 6.5       |
| 2      | NULL  | NULL      |
| 2      | 3.0   | NULL      |
| 3      | 4.0   | NULL      |
| 3      | NULL  | NULL      |
+-----+-----+-----+

-- If ORDER BY is not specified, rows from the first row to the last row belong to the current window.
The value of the third row is returned.
-- The second and third windows have only two rows.

select user_id, price, nth_value(price, 2) over
  (partition by user_id order by price) as nth_value from test_src;
+-----+-----+-----+
| user_id | price | nth_value |
+-----+-----+-----+
| 1      | 4.5   | NULL      | -- The current window has only one row. The second row exceeds the wi
ndow length.
| 1      | 5.5   | 5.5       |
| 1      | 5.5   | 5.5       |
| 1      | 6.5   | 5.5       |
| 2      | NULL  | NULL      |
| 2      | 3.0   | 3.0       |
| 3      | NULL  | NULL      |
| 3      | 4.0   | 4.0       |
+-----+-----+-----+

-- If ORDER BY is specified, rows from the first row to the current row belong to the current window.
The value of the second row is returned.

select user_id, price, nth_value(price, 1, true) over
  (partition by user_id) as nth_value from test_src;
+-----+-----+-----+
| user_id | price | nth_value |
+-----+-----+-----+
| 1      | 5.5   | 5.5       |

```

```

| 1 | 4.5 | 5.5 |
| 1 | 6.5 | 5.5 |
| 1 | 5.5 | 5.5 |
| 2 | NULL | 3.0 | -- The value of the first row is NULL, and therefore this row is skipped.
| 2 | 3.0 | 3.0 |
| 3 | 4.0 | 4.0 |
| 3 | NULL | 4.0 |
+-----+-----+-----+
-- If ORDER BY is not specified, rows from the first row to the last row belong to the current window.
-- The value of the first row is returned. skipNull is set to true.

```

3.15.5. Aggregate functions

Aggregate functions group multiple input records together to form a single output record. The input and output records have a many-to-one relationship. Aggregate functions can be used together with the **GROUP BY** clause in SQL.

aggregate function

COUNT

- Syntax

```
bigint count([distinct|all] value)
```

- Description

This function calculates the number of records that match the specified criteria.

- Parameters

- **distinct|all**: indicates whether to remove duplicate records during counting. The default value is **all**, which indicates that all records are counted. If it is set to **distinct**, only records with distinct values are counted.
- **value**: The value can be of any data type. If value is **NULL**, this row is not counted. value can be set to an asterisk (*****), namely, **count(*)** , which indicates that all rows are counted.

- Return value

A value of the **BIGINT** type is returned.

- Example

```
-- Assume that the tbla table has a COL1 column whose data type is BIGINT.
```

```
+-----+
```

```
| COL1 |
```

```
+-----+
```

```
| 1 |
```

```
+-----+
```

```
| 2 |
```

```
+-----+
```

```
| NULL |
```

```
+-----+
```

```
select count(*) from tbla; -- The return value is 3.
```

```
select count(col1) from tbla; -- The return value is 2.
```

Aggregate functions can be used together with the `GROUP BY` clause. For example, the `test_src` table has two columns: `key` is of the `STRING` type and `value` is of the `DOUBLE` type.

```
-- The data in this table is as follows:
```

```
+-----+-----+
```

```
| key | value |
```

```
+-----+-----+
```

```
| a | 2.0 |
```

```
+-----+-----+
```

```
| a | 4.0 |
```

```
+-----+-----+
```

```
| b | 1.0 |
```

```
+-----+-----+
```

```
| b | 3.0 |
```

```
+-----+-----+
```

```
-- Execute the following statement:
```

```
select key, count(value) as count from test_src group by key;
```

```
+-----+-----+
```

```
| key | count |
```

```
+-----+-----+
```

```
| a | 2 |
```

```
+-----+-----+
```

```
| b | 2 |
```

```
+-----+-----+
```

```
-- Aggregate functions aggregate values that have the same key. The usage of the following aggregate functions is the same as that of the COUNT function and therefore, is not detailed in this topic.
```

AVG

- Syntax

```
double avg(double value)
decimal avg(decimal value)
```

- Description

This function calculates the average value of a column.

- Parameters

value: The value must be of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to the DOUBLE type before calculation. If the input value is of another type, an error is returned. If value is NULL, this row is not calculated. Values of the BOOLEAN type are not calculated.

- Return value

If the input value is of the DECIMAL type, a value of the DECIMAL type is returned. If the input value is of another valid type, a value of the DOUBLE type is returned.

- Example

```
-- Assume that the tbla table has a value column whose data type is BIGINT.
+-----+
| value |
+-----+
| 1 |
| 2 |
| NULL |
+-----+

-- The average value of this column is as follows: (1 + 2)/2 = 1.5.
select avg(value) as avg from tbla;
+-----+
| avg |
+-----+
| 1.5 |
+-----+
```

MAX

- Syntax

```
max(value)
```

- Description

This function calculates the maximum value of a column.

- Parameters

value: The value can be of any data type. If value is NULL, this row is not calculated. Values of the BOOLEAN type are not calculated.

- Return value

The return value is of the same type as value.

- Example

```
-- Assume that the tbla table has a col1 column whose data type is BIGINT.
+-----+
| col1 |
+-----+
| 1 |
+-----+
| 2 |
+-----+
| NULL |
+-----+
select max(value) from tbla; -- The return value is 2.
```

MIN

- Syntax

```
MIN(value)
```

- Description

This function calculates the minimum value of a column.

- Parameters

value: The value can be of any data type. If value is NULL, this row is not calculated. Values of the BOOLEAN type are not calculated.

- Example

```
-- Assume that the tbla table has a value column whose data type is BIGINT.
+-----+
| value|
+-----+
| 1 |
+-----+
| 2 |
+-----+
| NULL |
+-----+
select min(value) from tbla; -- The return value is 1.
```

MEDIAN

- Syntax

```
double median(double number)
decimal median(decimal number)
```

- **Description**

This function calculates the median value of a column.

- **Parameters**

value: The value must be of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to the DOUBLE type before calculation. If the input value is of another type, an error is returned.

- **Return value**

A value of the DOUBLE or DECIMAL type is returned.

- **Example**

```
-- Assume that the tbla table has a value column whose data type is BIGINT.
+-----+
| value|
+-----+
| 1 |
+-----+
| 2 |
+-----+
| 3 |
+-----+
| 4 |
+-----+
| 5 |
+-----+
select MEDIAN(value) from tbla; -- The return value is 3.0.
```

STDDEV

- **Syntax**

```
double stddev(double number)
decimal stddev(decimal number)
```

- **Description**

This function calculates the population standard deviation of a column.

- **Parameters**

value: The value must be of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to the DOUBLE type before calculation. If the input value is of another type, an error is returned.

- Return value

A value of the DOUBLE or DECIMAL type is returned.

- Example

```
-- Assume that the tbla table has a value column whose data type is BIGINT.
+-----+
| value|
+-----+
| 1 |
+-----+
| 2 |
+-----+
| 3 |
+-----+
| 4 |
+-----+
| 5 |
+-----+

select STDDEV(value) from tbla; -- The return value is 1.4142135623730951.
```

STDDEV_SAMP

- Syntax

```
double stddev_samp(double number)
decimal stddev_samp(decimal number)
```

- Description

This function calculates the sample standard deviation of a column.

- Parameters

value: The value must be of the DOUBLE or DECIMAL type. If the input value is of the STRING or BIGINT type, it is implicitly converted to the DOUBLE type before calculation. If the input value is of another type, an error is returned.

- Return value

A value of the DOUBLE or DECIMAL type is returned.

- Example

```
-- Assume that the tbla table has a value column whose data type is BIGINT.

+-----+
| value|
+-----+
| 1 |
+-----+
| 2 |
+-----+
| 3 |
+-----+
| 4 |
+-----+
| 5 |
+-----+

select STDDEV_SAMP(value) from tbla; -- The return value is 1.5811388300841898.
```

SUM

- Syntax

```
sum(value)
```

- Description

This function calculates the sum of a column.

- Parameters

value: The value must be of the DOUBLE, DECIMAL, or BIGINT type. If the input value is of the STRING type, it is implicitly converted to the DOUBLE type before calculation. If value is NULL, this row is not calculated. Values of the BOOLEAN type are not calculated.

- Return value

If the input value is of the BIGINT type, a value of the BIGINT type is returned. If the input value is of the DOUBLE or STRING type, a value of the DOUBLE type is returned.

- Example

```
-- Assume that the tbla table has a value column whose data type is BIGINT.
```

```
+-----+
```

```
| value|
```

```
+-----+
```

```
| 1 |
```

```
+-----+
```

```
| 2 |
```

```
+-----+
```

```
| NULL |
```

```
+-----+
```

```
select sum(value) from tbla; -- The return value is 3.
```

WM_CONCAT

- Syntax

```
string wm_concat(string separator, string str)
```

- Description

This function uses the specified separator to connect values in str.

- Parameters

- separator: the delimiter, which is a constant of the STRING type. Constants of other types or non-constants result in exceptions.
- str: The value must be of the STRING type. If the input value is of the BIGINT, DOUBLE, or DATETIME type, it is implicitly converted to the STRING type before calculation. If the input value is of another type, an error is returned.

- Return value

A value of the STRING type is returned.

 **Note** If test_src in select wm_concat(',', name) from test_src; is an empty set, NULL is returned.

- Example

Merge values after grouping and sorting.

```
-- Create a table named test.
create table test(id int , alphabet string);
-- Insert data into the test table.
insert into test values (1,'a'),(1,'b'),(1,'c'),(2,'D'),(2,'E'),(2,'F');
-- Group and sort values in the table based on the id column. Then merge values in the same group.
SELECT id,WM_CONCAT(",alphabet) FROM test GROUP BY id ORDER BY id LIMIT 100;
```

id	_c1
1	abc
2	DEF

COLLECT_LIST

- Syntax

```
ARRAY collect_list(col)
```

- Description

This function aggregates the specified expressions in column col in a given group into an array.

- Parameters

col: a table column, which can be of any data type.

- Return value

A value of the ARRAY type is returned.

 **Note** If a new data type, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, needs to be used in a MaxCompute SQL statement, you must insert a SET statement before the SQL statement to enable the new data type.

- Session level: To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, plus commit and execute them together.
- Project level: You can enable a new data type at the project level. The project owner can run the following command to configure a project:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Other operations](#). For the precautions you must take when you enable data types at the project level, see [Date types](#).

COLLECT_SET

- Syntax

```
ARRAY collect_set(col)
```

- Description

This function aggregates the specified expressions in column `col` in a given group into an array without duplicate elements.

- Parameters

`col`: a table column, which can be of any data type.

- Return value

A value of the ARRAY type is returned.

 **Note** If a new data type, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, needs to be used in a MaxCompute SQL statement, you must insert a `SET` statement before the SQL statement to enable the new data type.

- Session level: To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, plus commit and execute them together.
- Project level: You can enable a new data type at the project level. The project owner can run the following command to configure a project:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Other operations](#). For the precautions you must take when you enable data types at the project level, see [Date types](#).

NUMERIC_HISTOGRAM

- Syntax

```
map<double, double> numeric_histogram(bigint buckets , double value)
```

- Description

This function returns the approximate histogram of a specific column.

- Parameters

- `buckets`: specifies the maximum number of buckets in the column whose approximate histogram is returned. The value must be of the BIGINT type.
- `value`: the column whose approximate histogram is to be returned. The value must be of the DOUBLE type.

- Return value

The return value is of the `MAP<DOUBLE,DOUBLE>` type. In the return value, a key indicates an x-coordinate, and its value indicates the height on the approximate histogram of the column.

PERCENTILE_APPROX

- Syntax

```
double percentile_approx(double col, p [, B])
array<double> percentile_approx(double col, array(p1 [, p2]...) [, B])
```

- Description

This function returns the approximate pth percentile of the column specified by col.

- Parameters

- p: the percentile value that ranges from 0.0 to 1.0. You can specify whether to return an approximate percentile value or an array that consists of percentile values.
- B: the accuracy of the return value. A high accuracy indicates a better approximate value. The default value is 10000. If the accuracy of values in column col is less than B, the exact percentile value is returned.

- Return value

`percentile_approx(double col, p [, B])` returns the approximate percentile value. `percentile_approx(double col, array(p1 [, p2]...) [, B])` returns an array consisting of percentile values.

- Example

```
SELECT percentile_approx(10.0, 0.5, 100); -- The return value is 10.0.
SELECT percentile_approx(10.0, array(0.5, 0.4, 0.1), 100); -- The return value is [10.0,10.0,10.0].
```

3.15.6. String functions

This topic describes the string functions provided by MaxCompute, which can be used in SQL statements.

CHAR_MATCHCOUNT

- Syntax

```
BIGINT CHAR_MATCHCOUNT(STRING str1, STRING str2)
```

- Description

This function returns the number of characters that belong to str1 and appear in str2.

- Parameters

str1 and str2: the valid UTF-8 strings. The function returns a negative value if invalid characters are found during the comparison of the two strings.

- Return value

Returns a value of the BIGINT type. NULL is returned if an input parameter is NULL.

- Examples

```
CHAR_MATCHCOUNT('abd','aabc') = 2
-- The a and b characters in str1 appear in str2.
```

CHR

- Syntax

```
STRING CHR(BIGINT ascii)
```

- Description

This function converts a specified ASCII code to the corresponding character.

- Parameters

ascii: a value of the BIGINT type, which indicates the ASCII code. Value range: 0 to 255. An exception is thrown if the input value is out of this range. The input value is implicitly converted to a value of the BIGINT type before computing if it is of the STRING, DOUBLE, or DECIMAL type. If the input value is of another data type, an exception is thrown.

- Return value

Returns a value of the STRING type. NULL is returned if the input parameter is NULL.

CONCAT

- Syntax

```
STRING CONCAT(STRING a, STRING b...)
```

- Description

This function concatenates all specified strings and returns the final string.

- Parameters

The values of all parameters are of the STRING type. The input value is implicitly converted to a value of the STRING type before computing if it is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type. If the input value is of another data type, an exception is thrown.

- Return value

Returns a value of the STRING type. NULL is returned if no input parameters are present or an input parameter is NULL.

- Examples

```
CONCAT('ab','c') = 'abc'  
CONCAT() = NULL  
CONCAT('a', null, 'b') = NULL
```

GET_JSON_OBJECT

- Syntax

```
STRING GET_JSON_OBJECT(STRING json,STRING path)
```

- Description

This function extracts the specified string from a standard JSON string based on `path`. The original data is read each time this function is called. Therefore, repeated calls may waste system resources and increase your cost. To avoid repeated calls, you can use the `GET_JSON_OBJECT` function together with User-Defined Table-Generating Functions (UDTFs). For more information, see [Convert JSON log data by using MaxCompute built-in functions and UDTFs](#).

- Parameters

- `json`: the string in the standard `JSON` format.
- `path`: the string that starts with the dollar sign (`$`), which describes the path of a JSON object in the JSON string. For more information about `path`, see [LanguageManual UDF](#).
 - `$`: indicates the root node.
 - `.`: indicates a child node.
 - `[]`: [number] indicates the array subscript. The array is represented in the format of `key[sub1][sub2][sub3].....`.
 - `*`: indicates the wildcard for `[]`, which returns the entire array. An asterisk (`*`) cannot be escaped.

- Return value

- Returns `NULL` if `json` is empty or invalid.
- Returns the corresponding string if `json` is valid and the specified `path` exists.
- A key cannot appear in an object twice. For example, `{a:1, a:0}` is not supported. Otherwise, the command may fail to be parsed.
- Emoji expressions are not supported.

- Example 1

```
+-----+
json
+-----+
{"store":
{"fruit":[{"weight":8,"type":"apple"}, {"weight":9,"type":"pear"}],
"bicycle":{"price":19.95,"color":"red"}
},
"email":"amy@only_for_json_udf_test.net",
"owner":"amy"
}
```

You can execute the following statements to extract data from the preceding `JSON` string.


```
odps> SELECT GET_JSON_OBJECT(src_json.json, '$.owner') FROM src_json;
amy
odps> SELECT GET_JSON_OBJECT(src_json.json, '$.store.fruit\[0]') FROM src_json;
{"weight":8,"type":"apple"}
odps> SELECT GET_JSON_OBJECT(src_json.json, '$.non_exist_key') FROM src_json;
NULL
```

Example 2

```
GET_JSON_OBJECT('{"array":["aaaa",1111],["bbbb",2222],["ccc",3333]}','$.array[1][1]')= "2222"
GET_JSON_OBJECT('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh[*]') = ["h0","h1","h2"]
GET_JSON_OBJECT('{"aaa":"bbb","ccc":{"ddd":"eee","fff":"ggg","hhh":["h0","h1","h2"]},"iii":"jjj"}','$.ccc.hhh[1]') = "h1"
```

INSTR

• Syntax

```
BIGINT INSTR(String str1, String str2[, BIGINT start_position[, BIGINT nth_appearance]])
```

• Description

This function returns the position of substring `str2` in string `str1`.

• Parameters

- `str1`: a value of the `STRING` type, which indicates the string to be searched. The input value is implicitly converted to a value of the `STRING` type before computing if it is of the `BIGINT`, `DOUBLE`, `DECIMAL`, or `DATETIME` type. If the input value is of another data type, an exception is thrown.
- `str2`: a value of the `STRING` type, which indicates the substring to be searched for. The input value is implicitly converted to a value of the `STRING` type before computing if it is of the `BIGINT`, `DOUBLE`, `DECIMAL`, or `DATETIME` type. If the input value is of another data type, an exception is thrown.
- `start_position`: a value of the `BIGINT` type. If it is of another data type, an exception is thrown. It indicates the character in `str1` from which the search will start. The start position defaults to the first character, marked as 1.
- `nth_appearance`: a value of the `BIGINT` type, which must be greater than 0. It indicates the `nth` time that the substring appears in the string, where `n` is specified by `nth_appearance`. An exception is thrown if the value of `nth_appearance` is of another data type or is not greater than 0.

• Return value

- Returns a value of the `BIGINT` type.
- Returns 0 if `str2` is not found in `str1`.
- Returns `NULL` if an input parameter is `NULL`.

- The matching always succeeds if `str2` is empty. For example, 1 is returned for `instr('abc', '')`.

- Examples

```
INSTR('Tech on the net', 'e') = 2
INSTR('Tech on the net', 'e', 1, 1) = 2
INSTR('Tech on the net', 'e', 1, 2) = 11
INSTR('Tech on the net', 'e', 1, 3) = 14
```

IS_ENCODING

- Syntax

```
BOOLEAN IS_ENCODING(String str, String from_encoding, String to_encoding)
```

- Description

This function determines whether the input string `str` can be converted from the character set specified by `from_encoding` to the character set specified by `to_encoding`. It can be used to determine whether the input string is garbled. Generally, `from_encoding` is set to utf-8 and `to_encoding` is set to gbk.

- Parameters

- `str` : a value of the STRING type. An empty string can belong to any character set.
- `from_encoding` and `to_encoding` : the source and target character sets. The values are of the STRING type.
- NULL is returned if an input parameter is NULL.

- Return value

Returns a value of the BOOLEAN type. If `str` can be converted, true is returned. Otherwise, false is returned.

KEYVALUE

- Syntax

```
KEYVALUE(String srcStr, String split1, String split2, String key)
KEYVALUE(String srcStr, String key) //split1 = ";", split2 = ":"
```

- Description

This function splits the source string `srcStr` into key-value pairs by `split1`, separates key-value pairs by `split2`, and then returns the value of the specified key.

- Parameters

- `srcStr` : the source string to be split.
- `key` : the key corresponds to the value returned after the source string is split by `split1` and `split2` in sequence. This parameter is of the STRING type.

- `split1` and `split2` : the strings that are used as delimiters to split the source string. If you do not specify these two parameters, `split1` defaults to `;` and `split2` defaults to `:`. If a key-value pair that is obtained after the source string is split by `split1` contains multiple delimiters specified by `split2`, the returned result is undefined.
- Return value
 - Returns a value of the STRING type.
 - Returns NULL if `split1` or `split2` is NULL.
 - Returns NULL if `scrStr` and `key` are NULL or if no `key` matches.
 - Returns the value corresponding to the first matched `key` if multiple key-value pairs match.
- Example 1

```
KEYVALUE('0:1\;1:2', 1) = '2'
```

The source string is `"0:1\;1:2"`. `split1` defaults to `;` and `split2` defaults to `:` because they are not specified.

After the source string is split by `split1`, the key-value pairs are `0:1\,1:2`. After the key-value pairs are separated by `split2`, the key-value pairs change to the following form:

```
0 1/
1 2
```

The value 2 corresponding to key 1 is returned.

Example 2

```
KEYVALUE("\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;",";",":","tf") = "21910" value:21910
```

The source string is `"\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;pf:0\;market:shoes\;instPayAmount:0\;"`. After the source string is split by `\;` specified by `split1`, the following key-value pairs are generated:

```
decreaseStore:1,xcard:1,isB2C:1,tf:21910,cart:1,shipping:2,pf:0,market:shoes,instPayAmount:0
```

After the key-value pairs are split by `:` specified by `split2`, the key-value pairs change to the following form:

```
decreaseStore 1
xcard 1
isB2C 1
tf 21910
cart 1
shipping 2
pf 0
market shoes
instPayAmount 0
```

The value 21910 corresponding to key `tf` is returned.

LENGTH

- Syntax

```
BIGINT LENGTH(String str)
```

- Description

This function returns the length of the string `str`.

- Parameters

- `str` : a value of the `STRING` type. The input value is implicitly converted to a value of the `STRING` type before computing if it is of the `BIGINT`, `DOUBLE`, `DECIMAL`, or `DATETIME` type. If the input value is of another data type, an exception is thrown.
- Returns a value of the `BIGINT` type. `NULL` is returned if the input parameter is `NULL`. -1 is returned if the input parameter is not encoded in UTF-8.

- Examples

```
LENGTH('hi! CN') = 6
```

LENGTHB

- Syntax

```
BIGINT LENGTHB(String str)
```

- Description

This function returns the length of the string `str` in bytes.

- Parameters

- `str` : a value of the `STRING` type. The input value is implicitly converted to a value of the `STRING` type before computing if it is of the `BIGINT`, `DOUBLE`, `DECIMAL`, or `DATETIME` type. If the input value is of another data type, an exception is thrown.
- Returns a value of the `BIGINT` type. `NULL` is returned if the input parameter is `NULL`.

- Examples

```
LENGTHB('hi! China!') = 10
```

MD5

- Syntax

```
STRING MD5(STRING value)
```

- Description

This function returns the MD5 value of the input string `value` .

- Parameters

`value` : a value of the STRING type. The input value is implicitly converted to a value of the STRING type before computing if it is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type. If the input value is of another data type, an exception is thrown.

- Return value

Returns a value of the STRING type. NULL is returned if the input parameter is NULL.

REGEXP_EXTRACT

- Syntax

```
STRING REGEXP_EXTRACT(STRING source, STRING pattern[, BIGINT occurrence])
```

- Description

This function splits the `source` string based on a given `pattern` and returns the characters in the `group` at the nth occurrence, where n is specified by `occurrence` .

- Parameters

- `source` : the string to split and search.
- `pattern` : a constant of the STRING type. An exception is thrown if `pattern` is an empty string or no `group` is specified in `pattern` .
- `occurrence` : a constant of the BIGINT type, which must be no less than 0. Otherwise, an exception is thrown. If you do not specify this parameter, it defaults to 1, indicating that characters in the first `group` are returned. If you set `occurrence` to 0, all substrings that match the regular expression specified by `pattern` are returned.

 **Note** Data is stored in the UTF-8 format. Chinese characters can be represented in hexadecimal. They are encoded in the range of [\x{4e00}-\x{9fa5}].

- Return value

Returns a value of the STRING type. Returns NULL if an input parameter is NULL.

REGEXP_INSTR

- Syntax

```
BIGINT REGEXP_INSTR(String source, String pattern[,  
BIGINT start_position[, BIGINT nth_occurrence[, BIGINT return_option]]])
```

- **Description**

This function returns the start or end position of the substring that matches `pattern` at the `nth` occurrence, where `n` is specified by `nth_occurrence`, in the `source` string from the start position specified by `start_position`.

- **Parameters**

- `source`: a value of the `STRING` type, which indicates the string to be searched.
- `pattern`: a constant of the `STRING` type. An exception is thrown if `pattern` is an empty string.
- `start_position`: a constant of the `BIGINT` type, which indicates the start position of a search. If you do not specify this parameter, its value defaults to 1. An exception is thrown if its value is of another data type or is no greater than 0.
- `nth_occurrence`: a constant of the `BIGINT` type. If you do not specify this parameter, its value defaults to 1, indicating the first occurrence of the matched substring. An exception is thrown if its value is of another data type or is no greater than 0.
- `return_option`: a constant of the `BIGINT` type, which can be 0 or 1. An exception is thrown if its value is of another data type or is out of the value range. 0 indicates that the start position of the matched substring is returned, and 1 indicates that the end position of the matched substring is returned.

- **Return value**

Returns a value of the `BIGINT` type. It is the start or end position of the matched substring in the `source` string based on the type specified by `return_option`. `NULL` is returned if an input parameter is `NULL`.

- **Examples**

```
REGEXP_INSTR("i love www.taobao.com", "o[[:alpha:]]{1}", 3, 2) = 14
```

REGEXP_REPLACE

- **Syntax**

```
String REGEXP_REPLACE(String source, String pattern, String replace_string[, BIGINT occurrence]  
)
```

- **Description**

This function substitutes the specified string `replace_string` for the substring that matches a given `pattern` at the `nth` occurrence, where `n` is specified by `occurrence`, in the `source` string, and returns the result.

- **Parameters**

- `source`: the string to be replaced.

- `pattern` : a constant of the `STRING` type, which indicates the pattern to be matched. An exception is thrown if `pattern` is an empty string.
- `replace_string` : a value of the `STRING` type, which is a string that is substituted for the substring matching the given `pattern` .
- `occurrence` : a constant of the `BIGINT` type, which must be no less than 0. It indicates that `replace_string` is substituted for the matching substring at the `nth` occurrence. The value 0 indicates that all matching substrings are replaced. An exception is thrown if its value is of another type or is less than 0. 0 is assumed by default.
- Return value
 - Returns a value of the `STRING` type. If the string to be replaced does not exist, the replacement does not occur.
 - Returns NULL if an input parameter is NULL.
 - Returns NULL if `replace_string` is NULL and certain substrings match the given `pattern` .
 - Returns the original string if `replace_string` is NULL but no substring matches the given `pattern` .
- Examples

```

REGEXP_REPLACE("123.456.7890", "([[:digit:]]{3})\\.[[:digit:]]{3}\\.[[:digit:]]{4})",
"((\\1)\\2-\\3", 0) = "(123)456-7890"
REGEXP_REPLACE("abcd", "(.)", "\\1 ", 0) = "a b c d "
REGEXP_REPLACE("abcd", "(.)", "\\1 ", 1) = "a bcd"
REGEXP_REPLACE("abcd", "(.)", "\\2", 1) = "abcd"
-- Only one group is defined in the pattern and the referenced second group does not exist.
-- Try to avoid this action. The result of referencing a nonexistent group is undefined.
REGEXP_REPLACE("abcd", "(. *)\\.($", "\\2", 0) = "d"
REGEXP_REPLACE("abcd", "a", "\\1", 0) = "bcd"
-- No group is defined in the pattern, so "\\1" references a nonexistent group.
-- Try to avoid this action. The result of referencing a nonexistent group is undefined.

```

REGEXP_SUBSTR

● Syntax

```

STRING REGEXP_SUBSTR(STRING source, STRING pattern[, BIGINT start_position[, BIGINT nth_occurrence]])

```

● Description

This function returns the string that matches a given `pattern` at the `nth` occurrence, where `n` is specified by `nth_occurrence` , in the `source` string from the start position specified by `start_position` .

● Parameters

- `source` : a value of the `STRING` type, which indicates the string to be searched.
 - `pattern` : a constant of the `STRING` type, which indicates the pattern to be matched. An exception is thrown if `pattern` is an empty string.
 - `start_position` : a constant of the `BIGINT` type, which must be greater than 0. An exception is thrown if its value is of another data type or is no greater than 0. If you do not specify this parameter, its value defaults to 1, indicating that the match starts from the first character of the `source` .
 - `nth_occurrence` : a constant of the `BIGINT` type, which must be greater than 0. An exception is thrown if its value is of another data type or is no greater than 0. If you do not specify this parameter, its value defaults to 1, indicating that the first matched substring is returned.
- **Return value**

Returns a value of the `STRING` type. `NULL` is returned if an input parameter is `NULL` or no substrings match.
 - **Examples**

```
REGEXP_SUBSTR("I love aliyun very much", "a[:alpha:]{5}") = "aliyun"  
REGEXP_SUBSTR('I have 2 apples and 100 bucks!', '[:blank:][:alnum:]*', 1, 1) = " have"  
REGEXP_SUBSTR('I have 2 apples and 100 bucks!', '[:blank:][:alnum:]*', 1, 2) = " 2"
```

REGEXP_COUNT

- **Syntax**

```
BIGINT REGEXP_COUNT(STRING source, STRING pattern[, BIGINT start_position])
```
- **Description**

This function returns the number of times a substring matches a given `pattern` in the `source` string from the start position specified by `start_position` .
- **Parameters**
 - `source` : a value of the `STRING` type, which indicates the string to be searched. An exception is thrown if its value is of another data type.
 - `pattern` : a constant of the `STRING` type, which indicates the pattern to be matched. An exception is thrown if `pattern` is an empty string or is of another data type.
 - `start_position` : a constant of the `BIGINT` type, which must be greater than 0. An exception is thrown if its value is of another data type or is no greater than 0. If you do not specify this parameter, its value defaults to 1, indicating that the match starts from the first character of the `source` string.
- **Return value**

Returns a value of the `BIGINT` type. 0 is returned if no substring is matched. Null is returned if an input parameter is `NULL`.
- **Examples**


```
REGEXP_COUNT('abababc', 'a.c') = 1  
REGEXP_COUNT('abcde', '[:alpha:]]{2}', 3) = 1
```

SPLIT_PART

- Syntax

```
STRING SPLIT_PART(String str, String separator, BIGINT start[, BIGINT end])
```

- Description

This function uses a delimiter specified by `separator` to split the string `str`, and returns a substring that starts from the character specified by `start` and ends with the character specified by `end`.

- Parameters

- `str`: a value of the `STRING` type, which indicates a string to be split. The input value is implicitly converted to a value of the `STRING` type before computing if it is of the `BIGINT`, `DOUBLE`, `DECIMAL`, or `DATETIME` type. If the input value is of another data type, an exception is thrown.
- `separator`: a constant of the `STRING` type, which is a separator used for splitting. It can be a character or string. If it is of another data type, an exception is thrown.
- `start`: a constant of the `BIGINT` type, which must be greater than 0. An exception is thrown if its value is not a constant or is of another data type. It indicates the start number of the segment to be returned, starting from 1. If `end` is not specified, the segment specified by `start` is returned.
- `end`: a constant of the `BIGINT` type, which must be no less than `start`. It indicates the end number of the segment to be returned. An exception is thrown if its value is not a constant or is of another data type. If `end` is not specified, the last segment is returned.

- Return value

- Returns a value of the `STRING` type.
- Returns an empty string if `start` is set to a value greater than the number of segments, for example, the string has 6 segments but the `start` value is greater than 6.
- Returns the entire string `str` if `separator` is absent in the string `str` and `start` is set to 1. An empty string is returned if `str` is an empty string.
- Returns the original string `str` if `separator` is an empty string.
- Returns the string between start and the last segment if `end` is set to a value greater than the number of segments.
- Returns `NULL` if an input parameter is `NULL`.

- Examples

```
SPLIT_PART('a,b,c,d', ',', 1) = 'a'
SPLIT_PART('a,b,c,d', ',', 1, 2) = 'a,b'
SPLIT_PART('a,b,c,d', ',', 10) = ''
```

SUBSTR

- Syntax

```
STRING SUBSTR(STRING str, BIGINT start_position[, BIGINT length])
```

- Description

This function returns a substring that starts from `start_position` in `str` and has a length specified by `length`.

- Parameters

- `str` : a value of the **STRING** type. The input value is implicitly converted to a value of the **STRING** type before computing if it is of the **BIGINT**, **DOUBLE**, **DECIMAL**, or **DATETIME** type. If the input value is of another data type, an exception is thrown.
- `start_position` : a value of the **BIGINT** type. 1 is assumed by default. An empty string is returned if `start_position` is 0. If `start_position` is a negative value, the start position is counted backwards from the last character of the string. For example, -1 indicates the last character, -2 indicates the second-to-last character, -3 indicates the third-from-last character, and so on. If the value is of another data type, an exception is thrown.
- `length` : a value of the **BIGINT** type, which indicates the length of the substring. Its value is greater than 0. An exception is thrown if its value is of another data type or is no greater than 0.

- Return value

Returns a value of the **STRING** type. **NULL** is returned if an input parameter is **NULL**.

 **Note** If `length` is not specified, the substring from the start position to the end of `str` is returned.

- Examples

```
SUBSTR("abc", 2) = "bc"
SUBSTR("abc", 2, 1) = "b"
SUBSTR("abc",-2,2) = "bc"
SUBSTR("abc",-3) = "abc"
```

SUBSTRING

- Syntax

```
STRING SUBSTRING(STRING|BINARY str, INT start_position[, INT length])
```

- Description

This function returns a substring that starts from `start_position` in `str` and has a length specified by `length`.

- Parameters

- `str` : a value of the STRING or BINARY type. NULL or an error is returned if the input value is of another data type.
- `start_position` : a value of the INT type. 1 is assumed by default. An empty string is returned if `start_position` is set to 0. If `start_position` is a negative value, the start position is counted backwards from the last character of the string. For example, -1 indicates the last character, -2 indicates the second-to-last character, -3 indicates the third-from-last character, and so on. If the value is of another data type, an exception is thrown.
- `length` : a value of the BIGINT type, which indicates the length of the substring. Its value is greater than 0. An exception is thrown if its value is of another data type or is no greater than 0.

- Return value

Returns a value of the STRING type. NULL is returned if an input parameter is NULL.

 **Note** If `length` is not specified, the substring from the start position to the end of `str` is returned.

- Examples

```
SUBSTRING('abc', 2) = 'bc'  
SUBSTRING('abc', 2, 1) = 'b'  
SUBSTRING('abc',-2,2) = 'bc'  
SUBSTRING('abc',-3,2) = 'ab'  
SUBSTRING(BIN(2345),2,3) = '001'
```

TOLOWER

- Syntax

```
STRING TOLOWER(STRING source)
```

- Description

This function converts the string `source` to a lowercase string.

- Parameters

`source` : a value of the STRING type. The input value is implicitly converted to a value of the STRING type before computing if it is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type. If the input value is of another data type, an exception is thrown.

- Return value

Returns a value of the STRING type. NULL is returned if the input parameter is NULL.

- Examples

```
tolower("aBcd") = "abcd"
tolower("ABCD") = "abcd"
```

TOUPPER

- Syntax

```
STRING TOUPPER(String source)
```

- Description

This function converts the string `source` to an uppercase string.

- Parameters

`source` : a value of the STRING type. The input value is implicitly converted to a value of the STRING type before computing if it is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type. If the input value is of another data type, an exception is thrown.

- Return value

Returns a value of the STRING type. NULL is returned if the input parameter is NULL.

- Examples

```
toupper("aBcd") = "ABCD"
toupper("abCd") = "ABCD"
```

TO_CHAR

- Syntax

```
STRING TO_CHAR(BOOLEAN value)
STRING TO_CHAR(BIGINT value)
STRING TO_CHAR(DOUBLE value)
STRING TO_CHAR(DECIMAL value)
```

- Description

This function converts data of the BOOLEAN, BIGINT, DECIMAL, or DOUBLE type to the STRING type.

- Parameters

`value` : The input value can be of the BOOLEAN, BIGINT, DECIMAL, or DOUBLE type. An exception is thrown if its value is of another data type.

- Return value

Returns a value of the STRING type. NULL is returned if the input parameter is NULL.

- Examples

```
TO_CHAR(123) = '123'  
TO_CHAR(true) = 'TRUE'  
TO_CHAR(1.23) = '1.23'  
TO_CHAR(null) = NULL
```

TRIM

- Syntax

```
STRING TRIM(STRING str)
```

- Description

This function eliminates the spaces on the left and right sides of the string `str`.

- Parameters

`str` : a value of the STRING type. The input value is implicitly converted to a value of the STRING type before computing if it is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type. If the input value is of another data type, an exception is thrown.

- Return value

Returns a value of the STRING type. NULL is returned if the input parameter is NULL.

LTRIM

- Syntax

```
STRING LTRIM(STRING str)
```

- Description

This function eliminates the spaces on the left side of the string `str`.

- Parameters

`str` : a value of the STRING type. The input value is implicitly converted to a value of the STRING type before computing if it is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type. If the input value is of another data type, an exception is thrown.

- Return value

Returns a value of the STRING type. NULL is returned if the input parameter is NULL.

- Examples

```
DELECT LTRIM(' abc ') FROM dual;  
-- Returned result:  
+-----+  
| _c0 |  
+-----+  
| abc |  
+-----+
```

RTRIM

- Syntax

```
STRING RTRIM(STRING str)
```

- Description

This function eliminates the spaces on the right side of the string `str`.

- Parameters

`str` : a value of the `STRING` type. The input value is implicitly converted to a value of the `STRING` type before computing if it is of the `BIGINT`, `DOUBLE`, `DECIMAL`, or `DATETIME` type. If the input value is of another data type, an exception is thrown.

- Return value

Returns a value of the `STRING` type. `NULL` is returned if the input parameter is `NULL`.

- Examples

```
SELECT RTRIM('a abc ') FROM dual;
-- Returned result:
+-----+
| _c0 |
+-----+
| a abc |
+-----+
```

REVERSE

- Syntax

```
STRING REVERSE(STRING str)
```

- Description

This function returns a string in reverse order.

- Parameters

`str` : a value of the `STRING` type. The input value is implicitly converted to a value of the `STRING` type before computing if it is of the `BIGINT`, `DOUBLE`, `DECIMAL`, or `DATETIME` type. If the input value is of another data type, an exception is thrown.

- Return value

Returns a value of the `STRING` type. `NULL` is returned if the input parameter is `NULL`.

- Examples

```
SELECT REVERSE('abcedfg') FROM dual;--Returned result:
```

```
+-----+
| _c0 |
+-----+
| gfdecba |
+-----+
```

REPEAT

- Syntax

```
STRING REPEAT(STRING str, BIGINT n)
```

- Description

This function returns `n` duplicates of the string `str`.

- Parameters

- `str` : a value of the STRING type. The input value is implicitly converted to a value of the STRING type before computing if it is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type. If the input value is of another data type, an exception is thrown.
- `n` : a value of the BIGINT type. The returned string cannot exceed 2 MB in length. An exception is thrown if `n` is NULL.

- Return value

Returns a value of the STRING type.

- Examples

```
SELECT REPEAT('abc',5) FROM lxw_dual; -- Return abcabcabcabcabc.
```

ASCII

- Syntax

```
BIGINT ASCII(STRING str)
```

- Description

This function returns the ASCII code of the first character in the string `str`.

- Parameters

`str` : a value of the STRING type. The input value is implicitly converted to a value of the STRING type before computing if it is of the BIGINT, DOUBLE, DECIMAL, or DATETIME type. If the input value is of another data type, an exception is thrown.

- Return value

Returns a value of the BIGINT type.

- Examples

```
SELECT ASCII('abcde') FROM dual; -- Return 97.
```

Additional functions of MaxCompute V2.0

MaxCompute V2.0 extends some mathematical functions. If an additional function that you use in an SQL statement involves new data types, you must insert the following `set` statement before the SQL statement:

```
set odps.sql.type.system.odps2=true;
```

 **Note** Add `set odps.sql.type.system.odps2=true;` before the SQL statement that uses the function, and commit and run it together with the SQL statement so that the new data types can be used.

This section describes new string functions in MaxCompute V2.0.

CONCAT_WS

- Syntax

```
STRING CONCAT_WS(STRING SEP, STRING a, STRING b...)  
STRING CONCAT_WS(STRING SEP, ARRAY)
```

- Description

This function concatenates all input strings in an array by using a specified delimiter.

- Parameters

- `SEP` : the delimiter of the `STRING` type. An exception is thrown if you do not specify this parameter.
- Parameters such as `a` and `b` : values of the `STRING` type. The input value is implicitly converted to a value of the `STRING` type before computing if it is of the `BIGINT`, `DOUBLE`, `DECIMAL`, or `DATETIME` type. If the input value is of another data type, an exception is thrown.

- Return value

Returns a value of the `STRING` type. `NULL` is returned if no input parameters are present or an input parameter is `NULL`.

- Examples

```
CONCAT_WS(':', 'name', 'hanmeimei')='name:hanmeimei'  
CONCAT_WS(':', 'avg', null, '34')=null
```

LPAD

- Syntax

```
STRING LPAD(STRING a, INT len, STRING b)
```


- Description

This function pads the left side of string `a` with string `b` until the new padded string has `len` characters.

- Parameters

- `len` : a value of the INT type.
- Parameters such as `a` and `b` : values of the STRING type.

- Return value

Returns a value of the STRING type. If `len` is smaller than the number of characters in `a`, `a` is truncated from the left to obtain a string with the number of characters specified by `len`. If `len` is 0, NULL is returned.

- Examples

```
lpad('abcdefgh',10,'12')='12abcdefgh'
lpad('abcdefgh',5,'12')='abcde'
lpad('abcdefgh',0,'12') -- Return NULL.
```

MaxCompute V2.0 provides additional date functions. If the functions you are using involve new data types, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, run the following SET statement to enable these data types:

- Session level: To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, and commit and execute them together.
- Project level: The project owner can set the project as needed. It takes 10 to 15 minutes for the settings to take effect. Run the following command:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Project operations](#). For the precautions you must take when you enable data types at the project level, see [Date types](#).

RPAD

- Syntax

```
STRING RPAD(STRING a, INT len, STRING b)
```

- Description

This function pads the right side of string `a` with string `b` until the new padded string has `len` characters.

- Parameters

- `len` : a value of the INT type.
- Parameters such as `a` and `b` : values of the STRING type.

- Return value

Returns a value of the STRING type. If `len` is smaller than the number of characters in `a`, `a` is truncated from the left to obtain a string with the number of characters specified by `len`. If `len` is 0, NULL is returned.

- Examples

```
rpadd('abcdefgh',10,'12')='abcdefgh12'  
rpadd('abcdefgh',5,'12')='abcde'  
rpadd('abcdefgh',0,'12') -- Return NULL.
```

MaxCompute V2.0 provides additional date functions. If the functions you are using involve new data types, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, run the following SET statement to enable these data types:

- Session level: To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, and commit and execute them together.
- Project level: The project owner can set the project as needed. It takes 10 to 15 minutes for the settings to take effect. Run the following command:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Project operations](#). For the precautions you must take when you enable data types at the project level, see [Date types](#).

REPLACE

- Syntax

```
STRING REPLACE(STRING a, STRING OLD, STRING NEW)
```

- Description

This function substitutes the string `NEW` for the part of string `a` that is exactly the same as the string `OLD`, and returns the string `a`.

- Parameters

The values of all parameters are of the STRING type.

- Return value

Returns a value of the STRING type. NULL is returned if an input parameter is NULL.

- Examples

```
REPLACE('ababab','abab','12')='12ab'  
REPLACE('ababab','cdf','123')='ababab'  
REPLACE('123abab456ab',null,'abab')=null
```

SOUNDEX

- Syntax

```
STRING SOUNDEX(STRING a)
```

- Description

This function converts a normal string to a string of the `soundex` type.

- Parameters

`a` : a value of the `STRING` type.

- Return value

Returns a value of the `STRING` type. `NULL` is returned if the input parameter is `NULL`.

- Examples

```
SOUNDEX('hello')='H400'
```

SUBSTRING_INDEX

- Syntax

```
STRING SUBSTRING_INDEX(STRING a, STRING sep, INT count)
```

- Description

This function truncates the string `a` to a substring from the first character to the `n`th delimiter, where `n` is specified by `count`. If `count` is a positive value, the string is truncated from left to right. Otherwise, the string is truncated from right to left.

- Parameters

`a` and `sep` are of the `STRING` type, whereas `count` is of the `INT` type.

- Return value

Returns a value of the `STRING` type. `NULL` is returned if an input parameter is `NULL`.

- Examples

```
SUBSTRING_INDEX('https://www.alibabacloud.com', '.', 2)='https://www.alibabacloud'
SUBSTRING_INDEX('https://www.alibabacloud.com', '.', -2)='alibabacloud.com'
SUBSTRING_INDEX('https://www.alibabacloud.com', null, 2)=null
```

MaxCompute V2.0 provides additional date functions. If the functions you are using involve new data types, such as `TINYINT`, `SMALLINT`, `INT`, `FLOAT`, `VARCHAR`, `TIMESTAMP`, or `BINARY`, run the following `SET` statement to enable these data types:

- Session level: To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, and commit and execute them together.
- Project level: The project owner can set the project as needed. It takes 10 to 15 minutes for the settings to take effect. Run the following command:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Project operations](#). For the precautions you must

take when you enable data types at the project level, see [Date types](#).

URL_ENCODE

- Syntax

```
STRING URL_ENCODE(STRING input[, STRING encoding])
```

- Description

This function encodes the input string in the `application/x-www-form-urlencoded` MIME format and returns the encoded string.

- All letters remain unchanged.
- Dots (.), hyphens (-), asterisks (*), and underscores (_) remain unchanged.
- Spaces are converted to plus signs (+).
- Other characters are converted to byte values based on the specified `encoding`. Each byte value is then represented in the format of `%xy`, where `xy` is the hexadecimal representation of the character value.

- Parameters

- `input`: the string to be encoded.
- `encoding`: the specified encoding format. GBK and UTF-8 are supported. If you do not specify this parameter, its value defaults to UTF-8.

- Return value

Returns a value of the STRING type. NULL is returned if the input parameter is NULL.

- Examples

```
URL_ENCODE('Example for URL_ENCODE:// dsf(fasfs)', 'GBK') = "Example+for+URL_ENCODE+%3A%2F%2F+dsf%28fasfs%29"
```

URL_DECODE

- Syntax

```
STRING URL_DECODE(STRING input[, STRING encoding])
```

- Description

This function converts an input string from the `application/x-www-form-urlencoded` MIME format to a normal string. This is the inverse function of `URL_ENCODE`.

- All letters remain unchanged.
- Dots (.), hyphens (-), asterisks (*), and underscores (_) remain unchanged.
- Each plus sign (+) is converted to a space.
- The `%xy` formatted sequence is converted to byte values. Consecutive byte values are decoded to the corresponding strings based on the input `encoding`.
- Other characters remain unchanged.

- The final return value of the function is a UTF-8 string.
- Parameters
 - `input` : the string to be decoded.
 - `encoding` : the specified encoding format. GBK and UTF-8 are supported. If you do not specify this parameter, its value defaults to UTF-8.

- Return value

Returns a value of the STRING type. NULL is returned if the input parameter is NULL.

- Examples

```
URL_DECODE('Example+for+URL_DECODE+%3A%2F%2F+dsf%28fasfs%29', 'GBK') = "Example for URL_D  
ECODE:// dsf(fasfs)"
```

JSON_TUPLE

- Syntax

```
STRING JSON_TUPLE(String json,String key1,String key2,...)
```

- Description

This function extracts strings from a standard JSON string based on specified keys.

- Parameters

- `json`: a value of the STRING type, which indicates a standard JSON string.
- `key`: a value of the STRING type, which is used to describe the path of a JSON object in the JSON string. . The value cannot start with a dollar sign (\$). You can enter multiple keys at a time.

- Return value

Returns a value of the STRING type.

 Note

- NULL is returned if json is empty or invalid.
- NULL is returned if key is empty, invalid, or does not exist in the JSON string.
- The corresponding string is returned if json is valid and key is also present.
- This function can parse JSON data that contains Chinese characters.
- This function can parse nested JSON data.
- This function can parse JSON data that contains nested arrays.
- The parsing action is equivalent to the execution of GET_JSON_OBJECT() along with `set odps.sql.udf.getjsonobj.new=true;`. To obtain multiple objects from a JSON string, you must call the GET_JSON_OBJECT() function multiple times. As a result, the JSON string is parsed multiple times. The JSON_TUPLE() function allows you to enter multiple keys at a time and the JSON string is parsed only once. Compared with GET_JSON_OBJECT(), JSON_TUPLE() is more efficient.
- JSON_TUPLE() is a User-Defined Table-Generating Function (UDTF). If you need to select other columns, use it along with LATERAL VIEW.

3.15.7. Other functions

This topic describes how to use functions such as CAST, DECODE, LEAST, ARRAY, SPLIT, and MAP.

other function DECODE CAST

CAST

Function declaration:

```
cast(expr as <type>)
```

Description: used to convert the result of an expression to another data type. For example, `cast('1' as bigint)` converts '1' of the STRING type to 1 of the INTEGER type. If the conversion fails, an error is returned.

Usage:

- `cast(double as bigint)` : converts a value of the DOUBLE type to the BIGINT type.
- `cast(string as bigint)` : converts a value of the STRING type to the BIGINT type. If the string is comprised of numerals expressed in the INTEGER form, it is converted to the BIGINT type. If the string is comprised of numerals expressed in the FLOAT or EXPONENTIAL form, it is converted to the DOUBLE type and then to the BIGINT type.
- The date format for `cast(string as datetime)` and `cast(datetime as string)` is yyyy-mm-dd hh:mi:ss.

COALESCE

Function declaration:

```
coalesce(expr1, expr2, ...)
```

Description: used to return the first non-NULL value in the list. If all values in the list are NULL, NULL is returned.

Parameters:

- `expr` : the value you want to test. All the values must be of the same data type or be NULL. Otherwise, an error is returned.
- At least one parameter is required. Otherwise, an error is returned.

Return value: The type of the return value must be the same as the input parameter.

DECODE

Function declaration:

```
decode(expression, search, result[, search, result]...[, default])
```

Description: used to implement the IF-THEN-ELSE conditional branching feature.

Parameters:

- `expression` : the expression that you want to compare.
- `search` : the search item that you want to compare with `expression` .
- `result` : the value returned if the values of `search` and `expression` match.
- `default` : the value returned if no search item matches the expression. If `default` is not specified, NULL is returned. This parameter is optional.

Return value:

- The matched `search` value is returned.
- If no matched item exists, `default` is returned.
- If `default` is not specified, NULL is returned.



Note

- At least three parameters are specified.
- The values of the `result` parameter must be of the same type or be NULL. Different data types cause an error. The values of `search` and `expression` must be of the same type. Otherwise, an error is returned.
- If `search` in the `DECODE` function has duplicated values that can match the expression, the first mapping result is returned.

Example:

```
select
decode(customer_id,
1, 'Taobao',
2, 'Alipay',
3, 'Aliyun',
Null, 'N/A',
'Others') as result
from sale_detail;
```

In this example, the `DECODE` function implements the `IF-THEN-ELSE` statement.

```
if customer_id = 1 then
result := 'Taobao';
elseif customer_id = 2 then
result := 'Alipay';
elseif customer_id = 3 then
result := 'Aliyun';
...
else
result := 'Others';
end if;
```

Note

- In normal cases, if "NULL = NULL" appears during the computing, the result returned by the MaxCompute SQL engine is NULL, while the `DECODE` function considers that the two NULL values are the same.
- In this example, if the value of `customer_id` is NULL, the `DECODE` function returns N/A.

GET_IDCARD_AGE

Function declaration:

```
get_idcard_age(idcardno)
```

Description: used to return an age based on the ID card number. The return value is the difference between the current year and the year of birth identified on the ID card.

Parameter:

`idcardno` : the ID card number of the `STRING` type. The value is a 15-digit or 18-digit number. During calculation, the validity of the ID card is checked based on the province code and the last digit of the ID card. NULL is returned if the check fails.

Return value: It is of the BIGINT type. If the input value is NULL, NULL is returned.

GET_IDCARD_BIRTHDAY

Function declaration:

```
get_idcard_birthday(idcardno)
```

Description: used to return the date of birth based on the ID card number.

Parameter:

`idcardno` : the ID card number of the STRING type. The value is a 15-digit or 18-digit number.

During calculation, the validity of the ID card is checked based on the province code and the last digit of the ID card. NULL is returned if the check fails.

Return value: It is of the DATETIME type. If the input value is NULL, NULL is returned.

GET_IDCARD_SEX

Function declaration:

```
get_idcard_sex(idcardno)
```

Description: used to return the gender based on the ID card number. The return value can be

`M` (male) or `F` (female).

Parameter:

`idcardno` : the ID card number of the STRING type. The value is a 15-digit or 18-digit number.

During calculation, the validity of the ID card is checked based on the province code and the last digit of the ID card. NULL is returned if the check fails.

Return value: It is of the STRING type. If the input value is NULL, NULL is returned.

GREATEST

Function declaration:

```
greatest(var1, var2, ...)
```

Description: used to return the maximum value of the input parameters.

Parameters:

`var1` and `var2` are input values. They can be of the BIGINT, DOUBLE, DECIMAL, DATETIME, or STRING type. If the values of all input parameters are NULL, NULL is returned.

Return value:

- The maximum value of the input parameters is returned. If no implicit conversion is required, the return value is of the same type as the input parameter.
- NULL is interpreted as the minimum value.
- If the input parameters are of different types, those of the DOUBLE, BIGINT, DECIMAL, and STRING types need to be converted to the DOUBLE type for comparison, and those of the

STRING and DATETIME types need to be converted to the DATETIME type for comparison. Implicit conversions of other types are not allowed.

- If `set odps.sql.hive.compatible` is set to `true` and the value of any input parameter is NULL, NULL is returned.

ORDINAL

Function declaration:

```
ordinal(BIGINT nth, var1, var2, ...)
```

Description: used to sort the input variables in ascending order, and return the value in the `nth` bit.

Parameters:

- `nth` : the position of the value to be returned. The value is of the BIGINT type. If it is set to NULL, NULL is returned.
- `var1` and `var2` : the input variables. They can be of the BIGINT, DOUBLE, DATETIME, or STRING type.

Return value:

- The value in the `nth` bit is returned. If no implicit conversion is required, the return value is of the same type as the input parameter.
- If a data type conversion is performed among the DOUBLE, BIGINT, and STRING types, a value of the DOUBLE type is returned. If a data type conversion is performed between the STRING and DATETIME types, a value of the DATETIME type is returned. Implicit conversions of other types are not allowed.
- NULL is interpreted as the minimum value.

Example:

```
ordinal(3, 1, 3, 2, 5, 2, 4, 6) = 2
```

PARTITION_EXISTS

Function declaration:

```
boolean partition_exists(string table_name, string... partitions)
```

Description: used to query whether a specific partition exists.

Parameters:

- `table_name`: the name of the table that you want to query. It is of the STRING type. You can specify a project name in the table name, for example, `my_proj.my_table`. If you do not specify a project name, the current project name is used by default.
- `partitions`: the name of the partition that you want to query. It is of the STRING type. The value of this parameter is a list of partitioning column values based on the sequence of partitioning columns in the table. The numbers of partitioning column values and partitioning

columns must be the same.

Return value: It is of the **BOOLEAN** type. If the specific partition exists, **True** is returned. Otherwise, **False** is returned.

Example:

```
-- Create partitioned table foo.  
CREATE TABLE foo (id BIGINT) PARTITIONED BY (ds STRING, hr STRING);  
  
-- Add a partition to foo.  
ALTER TABLE foo ADD PARTITION (ds='20190101', hr='1');  
  
-- Check whether partitions ds='20190101' and hr='1' exist.  
SELECT partition_exists('foo', '20190101', '1');
```

LEAST

Function declaration:

```
least(var1, var2, ...)
```

Description: used to return the minimum value of the input parameters.

Parameters:

`var1` and `var2` are input values. They can be of the **BIGINT**, **DOUBLE**, **DECIMAL**, **DATETIME**, or **STRING** type. If the values of all input parameters are **NULL**, **NULL** is returned.

Return value:

- The minimum value of the input parameters is returned. If no implicit conversion is required, the return value is of the same type as the input parameter.
- If a data type conversion is performed among the **DOUBLE**, **BIGINT**, and **STRING** types, a value of the **DOUBLE** type is returned. If a data type conversion is performed between the **STRING** and **DATETIME** types, a value of the **DATETIME** type is returned. If a data type conversion is performed among the **DECIMAL**, **DOUBLE**, **BIGINT**, and **STRING** types, a value of the **DECIMAL** type is returned. Implicit conversions of other types are not allowed.
- **NULL** is interpreted as the minimum value.

MAX_PT

Function declaration:

```
max_pt(table_full_name)
```

Description: used to return the maximum value of each first-level partition of a partitioned table in alphabetic order. The first-level partitions have data files.

Parameter:

`table_full_name` : the table name that has a project name, such as *prj.src*. It is of the **STRING** type. You must have the permissions to read the table.

Return value: The maximum value of each first-level partition is returned.

Example:

For example, `tbl` is a partitioned table that has the following partitions, and all the partitions have data files:

```
pt='20120901'
pt='20120902'
```

If you execute the following statement, the return value of `max_pt` is `'20120902'`, and the MaxCompute SQL statement reads data in the `pt='20120902'` partition:

```
select * from tbl where pt=max_pt('myproject.tbl');
```

Note

If a partition is added by using the `ALTER TABLE` statement and contains no data file, this partition is not returned.

UUID

Function declaration:

```
string TRING uuid()
```

Description: used to return a random ID, for example, `29347a88-1e57-41ae-bb68-a9edbdd94212`.

 **Note** The return value of this function is a random global ID that has a low probability of duplication.

SAMPLE

Function declaration:

```
boolean sample(x, y, column_name1, column_name2,...)
```

Description: used to sample all values read from `column_name` based on `x` and `y`, and filter out the rows that do not meet the sampling condition.

Parameters:

- `x` and `y`: indicate that the values fall into `x` portions by calling the hash function and the `y` th portion is used. The values of these parameters are of the BIGINT type. `y` is optional. If it is not specified, the first portion is used, and you must not specify `column_name`. `x` and `y` are integer constants, which are greater than 0. If they are of another type or their values are less than or equal to 0, an exception is reported. If `y` is greater than `x`, an exception is also reported. If the value of `x` or `y` is NULL, NULL is returned.

- `column_name` : the column you want to sample. This parameter is optional. If it is not specified, random sampling is performed based on the values of `x` and `y`. It can be of any type, and the column value can be NULL. No implicit conversion is performed. If `column_name` is the constant NULL, an error is reported.

Return value: It is of the BOOLEAN type.

 **Note** To avoid data skew due to the NULL value, uniform hashing is performed on the NULL values in `column_name` in `x` portions. If `column_name` is not specified and the data volume is small, the output is not necessarily uniform. In this case, we recommend that you specify `column_name` to obtain a uniform output.

Example:

Assume that the `tbla` table exists and has the `cola` column.

```
select * from tbla where sample (4, 1, cola) = true;
-- The values in cola fall into four portions by calling the hash function, and the first portion is used.
select * from tbla where sample (4, 2) = true;
-- The values in each row are randomly hashed to four portions, and the second portion is used.
```

CASE WHEN expressions

MaxCompute provides the following `CASE WHEN` syntax:

```
case value
when value1 then result1
when value2 then result2
...
else resultn
end
```

```
case
when (_condition1) then result1
when (_condition2) then result2
when (_condition3) then result3
...
else resultn
end
```

The `CASE WHEN` expression can return different values based on the result of the `VALUE` expression. The following example shows that different regions are obtained based on the value of `shop_name` :

```
select
case
when shop_name is null then 'default_region'
when shop_name like 'hang%' then 'zj_region'
end as region
from sale_detail;
```

Note

- If the types of all `result` values are only BIGINT and DOUBLE, the types are converted to the DOUBLE type and then values are returned.
- If `result` values of the STRING type exist, the types are converted to the STRING type and then values are returned. If the conversion fails, for example, the conversion from the BOOLEAN type to the STRING type, an error is returned.
- Conversions between other types are not allowed.

IF expressions

Syntax:

```
if(testCondition, valueTrue, valueFalseOrNull)
```

Description: used to determine whether `testCondition` is true. If it is true, `valueTrue` is returned. Otherwise, `valueFalse` or NULL is returned.

Parameters:

- `testCondition` : the expression that you want to check, which is of the BOOLEAN type.
- `valueTrue` : the value returned if `testCondition` is true.
- `valueFalseOrNull` : the value returned if `testCondition` is false. It can be set to NULL.

Return value: The type of the return value is the same as that of `valueTrue` or `valueFalseOrNull`.

Example:

```
select if(1=2,100,200) from dual;
-- The result is as follows:
+-----+
| _c0 |
+-----+
| 200 |
+-----+
```

SPLIT

Function declaration:

```
split(str, pat)
```

Description: used to return an array after `str` is split with `pat`.

Parameters:

- `str` : the string that you want to split, which is of the STRING type.
- `pat` : the delimiter of the STRING type, which supports regular expressions.

Return value: It is in the `array <STRING>` format, and contains `str` after being split with `pat`.

Example:

```
select split("a,b,c",",") from dual;
-- The result is as follows:
+-----+
| _c0 |
+-----+
| [a, b, c] |
+-----+
```

STR_TO_MAP

Function declaration:

```
str_to_map(text [, delimiter1 [, delimiter2]])
```

Description: used to split `text` into key-value pairs by using `delimiter1` and then separate keys from values in the key-values pairs by using `delimiter2`.

Parameters:

- `text` : the string that you want to split, which is of the STRING type.
- `delimiter1` : the delimiter used to split a string, which is of the STRING type. If it is not specified, a comma (,) is used by default.
- `delimiter2` : the delimiter used to separate keys from values, which is of the STRING type. If it is not specified, a period (.) is used by default.

Return value: It is in the `MAP<STRING, STRING>` format. The return value indicates that text is split by using `delimiter1`, and then by using `delimiter2`.

Example:

```
select str_to_map('test1&1-test2&2','-','&');
```

-- The result is as follows:

```
+-----+  
| a      |  
+-----+  
| {test1:1, test2:2} |
```

EXPLODE

Function declaration:

```
explode (var)
```

Description: used to transpose one row into multiple rows.

- If the parameter is of the ARRAY<T> type, the data of the ARRAY type stored in a column is converted to multiple-row data.
- If the parameter is of the MAP<K, V> type, each key-value pair of the MAP type stored in a column is transposed to a row with two columns, one for the key and the other for the value.

Parameter:

var : It can be of the ARRAY<T> or MAP<K, V> type.

Return value: Rows after the conversion are returned.

Note

Limits:

- A **SELECT** statement can only contain one **EXPLODE** function, and no other columns of a table are allowed.
- This function cannot be used with the **GROUP BY** , **CLUSTER BY** , **DISTRIBUTE BY** , or **SORT BY** clause.

Example:

```
explode(array(null, 'a', 'b', 'c')) col
```

MAP

Function declaration:

```
MAP map(K key1, V value1, K key2, V value2, ...)
```

Description: used to create mappings by using the given key-value pairs.

Parameters:

- All `key` types, including those after implicit conversions, must be the same and be of basic data types.
- All `value` types, including those after implicit conversions, must be the same and can be of any data types.

Return value: It is of the MAP type.

Example:

The field in the `t_table` table is `c1 BIGINT, c2 STRING, c3 STRING, c4 BIGINT, c5 BIGINT`. The data is described as follows:

```
+-----+-----+-----+-----+
| c1    | c2 | c3 | c4    | c5    |
+-----+-----+-----+-----+
| 1000  | k11| k21| 86    | 15    |
| 1001  | k12| k22| 97    | 2     |
| 1002  | k13| k23| 99    | 1     |
+-----+-----+-----+-----+
```

Execute the following statement:

```
select map(c2,c4,c3,c5) from t_table;
-- The result is as follows:
+-----+
| _c0 |
+-----+
| {k11:86, k21:15}|
| {k12:97, k22:2}|
| {k13:99, k23:1}|
+-----+
```

 **Note** If a new data type, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, needs to be used in a MaxCompute SQL statement, you must insert a `SET` statement before the SQL statement to enable the new data type.

- Session level: To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, plus commit and execute them together.
- Project level: You can enable a new data type at the project level. The project owner can run the following command to configure a project:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Other operations](#). For the precautions you must take when you enable data types at the project level, see [Date types](#).

MAP_KEYS

Function declaration:

```
ARRAY map_keys(map<K, V>)
```

Description: used to return all keys in the map parameter as an array.

Parameter: The parameter value is of the MAP type.

Return value: It is of the ARRAY type. If the input value is NULL, NULL is returned.

Example:

The field in the `t_table_map` table is `c1 BIGINT,t_map MAP<STRING,BIGINT>`. The data is described as follows:

```
+-----+-----+
| c1    | t_map |
+-----+-----+
| 1000   | {k11:86, k21:15}|
| 1001   | {k12:97, k22:2}|
| 1002   | {k13:99, k23:1}|
+-----+-----+
```

Execute the following statement:

```
select c1,map_keys(t_map) from t_table_map;
```

-- The result is as follows:

```
+-----+-----+
| c1    | _c1 |
+-----+-----+
| 1000   | [k11, k21]|
| 1001   | [k12, k22]|
| 1002   | [k13, k23]|
+-----+-----+
```

MAP_VALUES

Function declaration:

```
ARRAY map_values(map<K, V>)
```

Description: used to return all values in the map parameter as an array.

Parameter: The parameter value is of the MAP type.

Return value: It is of the ARRAY type. If the input value is NULL, NULL is returned.

Example:

```
select map_values(map('a',123,'b',456));
-- The result is as follows:
[123, 456]
```

ARRAY

Function declaration:

```
ARRAY array(value1,value2, ...)
```

Description: used to create an array by using input values.

Parameters: The parameters can be of any type, but the types of all parameters must be the same.

Return value: It is of the ARRAY type.

Example:

The field in the `t_table` table is `c1 BIGINT, c2 STRING, c3 STRING, c4 BIGINT, c5 BIGINT`. The data is described as follows:

```
+-----+-----+-----+-----+
| c1    | c2 | c3 | c4    | c5    |
+-----+-----+-----+-----+
| 1000  | k11 | k21 | 86    | 15    |
| 1001  | k12 | k22 | 97    | 2     |
| 1002  | k13 | k23 | 99    | 1     |
+-----+-----+-----+-----+
```

Execute the following statement:

```
select array(c2,c4,c3,c5) from t_table;
-- The result is as follows:
+-----+
| _c0 |
+-----+
| [k11, 86, k21, 15] |
| [k12, 97, k22, 2] |
| [k13, 99, k23, 1] |
+-----+
```

 **Note** If a new data type, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, needs to be used in a MaxCompute SQL statement, you must insert a `SET` statement before the SQL statement to enable the new data type.

- **Session level:** To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, plus commit and execute them together.
- **Project level:** You can enable a new data type at the project level. The project owner can run the following command to configure a project:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Other operations](#). For the precautions you must take when you enable data types at the project level, see [Date types](#).

SIZE

Function declaration:

```
INT size(map)
INT size(array)
```

Description: `size(map)` is used to return the number of key-value pairs in the specified `map` parameter. `size(array)` is used to return the number of elements in the specified `array` parameter.

Parameters:

- `map`: the data of the MAP type.
- `array`: the data of the ARRAY type.

Return value: It is of the INT type.

Example:

```
select size(map('a',123,'b',456)) from dual; -- Value 2 is returned.
select size(map('a',123,'b',456,'c',789)) from dual; -- Value 3 is returned.
select size(array('a','b')) from dual; -- Value 2 is returned.
select size(array(123,456,789)) from dual; -- Value 3 is returned.
```

ARRAY_CONTAINS

Function declaration:

```
boolean array_contains(ARRAY<T> a,value v)
```

Description: used to check whether array `a` contains value `v`.

Parameters:

- **a** : the data of the ARRAY type.
- **v** : The value must be of the same type as the data type in the array.

Return value: It is of the BOOLEAN type.

Example:

The field in the `t_table_array` table is `c1 BIGINT, t_array ARRAY<STRING>`. The data is described as follows:

```
+-----+-----+
| c1    | t_array |
+-----+-----+
| 1000  | [k11, 86, k21, 15] |
| 1001  | [k12, 97, k22, 2] |
| 1002  | [k13, 99, k23, 1] |
+-----+-----+
```

Execute the following statement:

```
select c1, array_contains(t_array,'1') from t_table_array;
```

-- The result is as follows:

```
+-----+-----+
| c1    | _c1 |
+-----+-----+
| 1000  | false |
| 1001  | false |
| 1002  | true  |
+-----+-----+
```

SORT_ARRAY

Function declaration:

```
ARRAY sort_array(ARRAY<T>)
```

Description: used to sort a given array.

Parameter:

`ARRAY<T>` : the data of the ARRAY type. Data in the array can be of any type.

Return value: It is of the ARRAY type.

Example:

The field in the `t_array` table is `c1 ARRAY<STRING>,c2 ARRAY<INT>,c3 ARRAY<STRING>`. The data is described as follows:

```

+-----+-----+-----+
| c1      | c2      | c3      |
+-----+-----+-----+
| [a, c, f, b] | [4, 5, 7, 2, 5, 8] | [You, Me, Him] |
+-----+-----+-----+

```

Execute the following statement:

```

select sort_array(c1),sort_array(c2),sort_array(c3) from t_array;
-- The result is as follows:
[a, b, c, f] [2, 4, 5, 5, 7, 8] [Him, You, Me]

```

 **Note** If a new data type, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, needs to be used in a MaxCompute SQL statement, you must insert a **SET** statement before the SQL statement to enable the new data type.

- **Session level:** To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, plus commit and execute them together.
- **Project level:** You can enable a new data type at the project level. The project owner can run the following command to configure a project:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Other operations](#). For the precautions you must take when you enable data types at the project level, see [Date types](#).

POSEXPLODE

Function declaration:

```
posexplode(ARRAY<T>)
```

Description: used to convert the array into a table that has two columns. The first column lists subscripts of each value in the array, starting from 0, and the second column lists array elements.

Parameter:

ARRAY<T> : the data of the ARRAY type. Data in the array can be of any type.

Return value: The generated table is returned.

Example:

```
select posexplode(array('a','c','f','b')) from dual;
```

-- The result is as follows:

```
+-----+-----+
| pos | val |
+-----+-----+
| 0 | a |
| 1 | c |
| 2 | f |
| 3 | b |
+-----+-----+
```

 **Note** If a new data type, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, needs to be used in a MaxCompute SQL statement, you must insert a **SET** statement before the SQL statement to enable the new data type.

- Session level: To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, plus commit and execute them together.
- Project level: You can enable a new data type at the project level. The project owner can run the following command to configure a project:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Other operations](#). For the precautions you must take when you enable data types at the project level, see [Date types](#).

STRUCT

Function declaration:

```
STRUCT struct(value1,value2, ...)
```

Description: used to create a struct based on the given `value` list.

Parameter:

`value` : It can be of any type.

Return value: It is of the STRUCT type. The fields are consecutively named, for example, `col1`, `col2`, and so on.

Example:

```
select struct('a',123,'ture',56.90) from dual;
-- The result is as follows:
{col1:a, col2:123, col3:true, col4:56.9}
```

 **Note** If a new data type, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, needs to be used in a MaxCompute SQL statement, you must insert a `SET` statement before the SQL statement to enable the new data type.

- **Session level:** To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, plus commit and execute them together.
- **Project level:** You can enable a new data type at the project level. The project owner can run the following command to configure a project:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Other operations](#). For the precautions you must take when you enable data types at the project level, see [Date types](#).

NAMED_STRUCT

Function declaration:

```
STRUCT named_struct(string name1, T1 value1, string name2, T2 value2, ...)
```

Description: used to create a struct based on the given `name/value` list.

Parameters:

- `value`: It can be of any type.
- `name`: the name of the field of the STRING type.

Return value: It is of the STRUCT type. The fields are consecutively named, for example, `col1`, `col2`, and so on.

Example:

```
select named_struct('user_id',10001,'user_name','LiLei','married','F','weight',63.50) from dual;
-- The result is as follows:
{user_id:10001, user_name:LiLei, married:F, weight:63.5}
```


 **Note** If a new data type, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, needs to be used in a MaxCompute SQL statement, you must insert a `SET` statement before the SQL statement to enable the new data type.

- **Session level:** To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, plus commit and execute them together.
- **Project level:** You can enable a new data type at the project level. The project owner can run the following command to configure a project:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Other operations](#). For the precautions you must take when you enable data types at the project level, see [Date types](#).

INLINE

Function declaration:

```
inline(array<struct<f1:T1, f2:T2, ... >>)
```

Description: used to expand a given struct array with each array element corresponding to a row and each struct element corresponding to a column in each row.

Parameter:

struct : The values in the array can be of any type.

Return value: The generated table is returned.

Example:

The field in the `t_table` table is `t_struct`

`struct<user_id:bigint,user_name:string,married:string,weight:double>`. The data is described as follows:

```
+-----+
| t_struct |
+-----+
| {user_id:10001, user_name:LiLei, married:N, weight:63.5} |
| {user_id:10002, user_name:HanMeiMei, married:Y, weight:43.5} |
+-----+
```

Execute the following statement:

```
select inline(array(t_struct)) from t_table;
-- The result is as follows:
+-----+-----+-----+-----+
| user_id | user_name | married | weight |
+-----+-----+-----+-----+
| 10001   | LiLei    | N       | 63.5   |
| 10002   | HanMeiMei | Y      | 43.5   |
+-----+-----+-----+-----+
```

 **Note** If a new data type, such as TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, or BINARY, needs to be used in a MaxCompute SQL statement, you must insert a **SET** statement before the SQL statement to enable the new data type.

- **Session level:** To use a new data type, you must insert `set odps.sql.type.system.odps2=true;` before the SQL statement, plus commit and execute them together.
- **Project level:** You can enable a new data type at the project level. The project owner can run the following command to configure a project:

```
setproject odps.sql.type.system.odps2=true;
```

For more information about `setproject`, see [Other operations](#). For the precautions you must take when you enable data types at the project level, see [Date types](#).

TABLE_EXISTS

Function declaration:

```
boolean table_exists(string table_name)
```

Description: used to query whether a specific table exists.

Parameter:

table_name: the name of the table that you want to query. It is of the STRING type. You can specify a project name in the table name, for example, `my_proj.my_table`. If you do not specify a project name, the current project name is used by default.

Return value: It is of the BOOLEAN type. If the specified table exists, True is returned. Otherwise, False is returned.

Example:

```
-- Used in the SELECT statement.
SELECT IF(table_exists('abd'), col1, col2) FROM src;
```

TRANS_ARRAY

Function declaration:

```
trans_array (num_keys, separator, key1,key2,...,col1, col2,col3) as (key1,key2,...,col1, col2)
```

Description: used to transpose one row into multiple rows, specifically, transpose an array separated by fixed delimiters in a column into multiple rows.

Parameters:

- **num_keys** : The value is a constant of the BIGINT type, which must be greater than or equal to 0. This parameter specifies the number of columns that can be used as **key** when you transpose one row into multiple rows.
- **key** : duplicate columns in multiple rows when you transpose one row into multiple rows.
- **separator** : the separator used to separate a string into multiple elements. It is a constant of the STRING type. If it is left blank, an error is returned.
- **keys** : columns that are used as **key** during the transposition. The number of keys is specified by **num_keys** . If **num_keys** equals the total number of all columns (which means that all columns are used as **key**), only one row is returned.
- **cols** : the array to be converted to rows. All columns following **keys** are considered as arrays to be transposed. The parameter value must be of the STRING type to store arrays in the STRING format, such as `Hangzhou;Beijing;shanghai` . The values in this array are separated with semicolons (;).

Return value: The rows after the transposition are returned. The names of new columns are specified by **as** . The data types of columns used as the **key** remain unchanged. All other columns are of the STRING type. The number of separated rows depends on the array with the maximum number. If the number of rows is insufficient, NULL is added.

 Note**UDTF limits:**

- All columns that are used as **key** must be placed before the columns to be transposed.
- Only one UDTF is allowed in a **SELECT** statement.
- This function cannot be used with the **GROUP BY** , **CLUSTER BY** , **DISTRIBUTE BY** , or **SORT BY** clause.

Example:

- Data in the **t_table** table is described as follows:

```

+-----+-----+-----+
| login_id | login_ip | login_time |
+-----+-----+-----+
| wangwangA | 192.168.0.1,192.168.0.2 | 20120101010000,20120102010000 |
| wangwangB | 192.168.45.10,192.168.67.22,192.168.6.3 | 20120111010000,20120112010000,20120223080000 |
+-----+-----+-----+

```

Execute the following statement:

```
select trans_array(1, "", login_id, login_ip, login_time) as (login_id,login_ip,login_time) from t_table;
```

The result is as follows:

```

+-----+-----+-----+
| login_id | login_ip | login_time |
+-----+-----+-----+
| wangwangB | 192.168.45.10 | 20120111010000 |
| wangwangB | 192.168.67.22 | 20120112010000 |
| wangwangB | 192.168.6.3 | 20120223080000 |
| wangwangA | 192.168.0.1 | 20120101010000 |
| wangwangA | 192.168.0.2 | 20120102010000 |
+-----+-----+-----+

```

- The table contains the following data:

```

Login_id LOGIN_IP LOGIN_TIME
wangwangA 192.168.0.1,192.168.0.2 20120101010000

```

NULL is added to complete the array which has insufficient data.

```

Login_id Login_ip Login_time
wangwangA 192.168.0.1 20120101010000
wangwangA 192.168.0.2 NULL

```

TRANS_COLS

Function declaration:

```
trans_cols (num_keys, key1,key2,...,col1, col2,col3) as (idx, key1,key2,...,col1, col2)
```

Description: used to transpose one row into multiple rows, specifically, transpose different columns into different rows.

Parameters:

- **num_keys** : The value is a constant of the BIGINT type, which must be greater than or equal to 0. This parameter specifies the number of columns that can be used as **key** when you

transpose one row into multiple rows.

- **key** : duplicate columns in multiple rows when you transpose one row to multiple rows.
- **separator** : the separator used to separate a string into multiple elements. It is a constant of the **STRING** type. If it is left blank, an error is returned.
- **keys** : columns that are used as **key** during the transposition. The number of keys is specified by **num_keys** . If **num_keys** equals the total number of all columns (which means that all columns are used as **key**), only one row is returned.
- **cols** : the columns to be converted into rows. The columns must be of the same type.

Return value: The rows after the transposition are returned. The names of new columns are specified by **as** . The first output column is the subscripts of the transposition, which start with 1. The data types of the columns used as keys remain unchanged, while the data types of other columns are the same as the originals.

Note

UDTF limits:

- All columns that are used as **key** must be placed before the columns to be transposed.
- Only one UDTF is allowed in a **SELECT** statement.
- This function cannot be used with the **GROUP BY** , **CLUSTER BY** , **DISTRIBUTE BY** , or **SORT BY** clause.

Example:

Data in the **t_table** table is described as follows:

```
+-----+-----+-----+
| Login_id | Login_ip1 | Login_ip2 |
+-----+-----+-----+
| wangwangA | 192.168.0.1 | 192.168.0.2 |
+-----+-----+-----+
```

Execute the following statement:

```
select trans_cols(1, login_id, login_ip1, login_ip2) as (idx, login_id, login_ip) from t_table;
```

The result is as follows:

```
idx  login_id  login_ip
1   wangwangA  192.168.0.1
2   wangwangA  192.168.0.2
```

3.16. UDF

3.16.1. Overview

MaxCompute offers a variety of built-in functions to meet your computing requirements. You can also use user-defined functions (UDFs).

UDF

In the broad sense, UDFs include user-defined scalar functions (UDFs), user-defined aggregation functions (UDAFs), and user-defined table-valued functions (UDTFs). In the narrow sense, UDFs refer only to user-defined scalar functions. The term UDF is frequently used in Alibaba Cloud documentation. You can determine whether it is used in the narrow sense based on the context.

UDFs work in a way similar to [built-in functions](#). For the mapping between Java and MaxCompute data types, see [Java UDF](#).

Note If a UDF has the same name as a built-in function, MaxCompute executes the UDF by default. To call the built-in function, you must execute the `select ::Function name (expression) ; statement`. For example, if a CONCAT UDF and a CONCAT built-in function are available in MaxCompute, MaxCompute calls the UDF rather than the built-in function. To call the CONCAT built-in function, execute the `select ::concat('ab','c') ; statement`.

UDF types

The following table describes three types of UDFs supported by MaxCompute.

UDF type	Description
UDF	User-defined scalar function. The relationship between the input and output of a user-defined scalar function is one-to-one mapping. One value is returned each time a user-defined scalar function reads one row of data.
UDTF	A user-defined table-valued function. It is used in scenarios where multiple rows of data are returned after you call each function. Only this type of function can return multiple fields. A UDTF is not equivalent to a user-defined type (UDT).
UDAF	A user-defined aggregation function. The relationship between the input and output of a user-defined aggregation function is many-to-one mapping. Multiple input records are aggregated to generate one output value. It can be used with the GROUP BY clause of SQL. For more information, see Aggregate functions .

Note When you use a UDF in an SQL statement, the system may prompt insufficient memory. The reason is that the memory size required for the computing task exceeds the default memory size. In this case, you can execute the `set odps.sql.udf.joiner.jvm.memory=xxxx;` statement to adjust the memory size.

MaxCompute UDFs can be shared by multiple projects. UDFs in one project can be used in another project. The SQL statement for cross-project UDF sharing is `select other_project:udf_in_other_project(arg0, arg1) as res from table_t;` . For more information, see [Authorize users](#).

UDF SDK dependency installation

If you use [Maven](#), you can search for odps-sdk-udf in the [Maven repository](#) to obtain your required SDK for Java. The configuration information is as follows:

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-udf</artifactId>
  <version>0.29.10-public</version>
</dependency>
```

3.16.2. Java UDF

MaxCompute supports user-defined functions (UDFs), user-defined aggregate functions (UDAFs), and user-defined table-valued functions (UDTFs). This topic describes how to implement the three types of functions by using Java.

Data type

Java UDFs in MaxCompute V2.0 support more basic data types in addition to BIGINT, STRING, DOUBLE, and BOOLEAN. They also support complex data types such as ARRAY, MAP, and STRUCT, as well as Writable types.

- To use a new basic data type, Java UDFs specify their function signatures in the following ways:
 - A UDTF uses the `@Resolve` annotation, for example, `@Resolve("smallint->varchar(10)")` .
 - A UDF uses the `evaluate` method. In this case, MaxCompute built-in function types and Java function types are one-to-one mapping.
 - A UDAF uses the `@Resolve` annotation. MaxCompute V2.0 supports the use of new basic data types in annotations, for example, `@Resolve("smallint->varchar(10)")` .
- To use a complex data type, Java UDFs specify their function signatures in the following ways:
 - A UDTF uses the `@Resolve` annotation, for example, `@Resolve("array<string>,struct<a1:bigint,b1:string>,string->map<string,bigint>,struct<b1:bigint>")` .
 - A UDF uses the signature of the `evaluate` method to match input and output types. For more information, see the mapping between MaxCompute and Java data types. The following examples show the mapping of data types supported by MaxCompute and Java:
 - ARRAY in MaxCompute is equivalent to `java.util.List` .
 - MAP in MaxCompute is equivalent to `java.util.Map` .
 - STRUCT in MaxCompute is equivalent to `com.aliyun.odps.data.Struct` .

- A UDAF uses the `@Resolve` annotation. MaxCompute V2.0 supports the use of new data types in annotations, for example, `@Resolve("smallint->varchar(10)")`.

 Note

- You can use `type,*` to pass any number of parameters, for example, `@resolve("string,*->array<string>")`. Note that you must add Subtype after ARRAY.
- The field name and field type of `com.aliyun.odps.data.Struct` cannot be reflected. Therefore, you must use `@Resolve` to obtain them. In other words, to use STRUCT in a UDF, you must add the `@Resolve` annotation to the UDF class. This annotation only affects the overloads of parameters or return values that contain `com.aliyun.odps.data.Struct`.
- Only one `@Resolve` annotation can be provided to the class. Therefore, only one overload with a STRUCT parameter or return value exists in a UDF.

Mapping between MaxCompute and Java data types

- The following table describes the mapping between MaxCompute and Java data types.

MaxCompute Type	Java 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
STRING	java.lang.String
VARCHAR	com.aliyun.odps.data.Varchar
BINARY	com.aliyun.odps.data.Binary
DATETIME	java.util.Date
TIMESTAMP	java.sql.Timestamp
ARRAY	java.util.List
MAP	java.util.Map

MaxCompute Type	Java Type
STRUCT	com.aliyun.odps.data.Struct

 Note

- Make sure that the input and output parameters in a UDF are of a Java type. Otherwise, error ODPS-0130071 is returned.
- Java data types and the data types of returned values are objects and must start with an uppercase letter.
- The NULL value in SQL statements is represented by a NULL reference in Java. Java Primitive Type cannot have NULL values and therefore must not be used.
- The ARRAY type in the preceding table maps the LIST data type in Java.

- MaxCompute V2.0 allows you to use Writable types as parameters and return values when you define Java UDFs. The following table describes the mapping between MaxCompute data types and Java Writable types.

MaxCompute Type	Java Writable Type
TINYINT	ByteWritable
SMALLINT	ShortWritable
INT	IntWritable
BIGINT	LongWritable
FLOAT	FloatWritable
DOUBLE	DoubleWritable
DECIMAL	BigDecimalWritable
BOOLEAN	BooleanWritable
STRING	Text
VARCHAR	VarcharWritable
BINARY	BytesWritable
DATETIME	DatetimeWritable
TIMESTAMP	TimestampWritable
INTERVAL_YEAR_MONTH	IntervalYearMonthWritable
INTERVAL_DAY_TIME	IntervalDayTimeWritable
ARRAY	N/A

MaxCompute Type	Java Writable Type
MAP	N/A
STRUCT	N/A

UDF

UDFs must inherit the class `com.aliyun.odps.udf.UDF` and use the `evaluate` method of the class. The `evaluate` method must be a non-static public method and its parameter and return value are used as the UDF signature in SQL statements. You can implement multiple `evaluate` methods in a UDF. To call a UDF, the framework matches the correct `evaluate` method based on the parameter type called by the UDF.

Note

- We recommend that you do not use classes that have the same class name but different function logic in different JAR packages. For example, in `UDF(UDAF/UDTF): udf1, udf2`, the JAR package of `udf1` is `udf1.jar` and the JAR package of `udf2` is `udf2.jar`. Assume that both JAR packages contain the `com.aliyun.UserFunction.class` class. If you use two UDFs in the same SQL statement at the same time, the system randomly loads either of them and may cause UDF execution inconsistency or even compilation failure.
- If you execute an SQL statement that has two UDFs, the UDFs are not isolated. Both UDFs share the same classpath. If the resources referenced by the UDFs contain the same class, the class that classloader attempts to load is uncertain. To avoid this issue, make sure that the resources referenced by two UDFs do not contain the same class.

You can use `void setup(ExecutionContext ctx)` to initialize a UDF and `void close()` to terminate a UDF.

The method of using UDFs is the same as that of using built-in functions of MaxCompute SQL. For more information, see [Mathematical functions](#).

The following example shows how to implement a UDF:

```
package org.alidata.odps.udf.examples;
import com.aliyun.odps.udf.UDF;

public final class Lower extends UDF {
    public String evaluate(String s) {
        if (s == null) {
            return null;
        }
        return s.toLowerCase();
    }
}
```

The following example shows how to implement Concat by using the Writable type:

```
package com.aliyun.odps.udf.example;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF;
public class MyConcat extends UDF {
    private Text ret = new Text();
    public Text evaluate(Text a, Text b) {
        if (a == null || b == null) {
            return null;
        }
        ret.clear();
        ret.append(a.getBytes(), 0, a.getLength());
        ret.append(b.getBytes(), 0, b.getLength());
        return ret;
    }
}
```

 Note

The package containing all Writable types is `com.aliyun.odps.io`. To use Writable types, you can download the `odps-sdk-commons` package from the [API document website](#).

The following table describes the SDK packages provided by MaxCompute.

Package name	Description
<code>odps-sdk-core</code>	Provides basic functions of MaxCompute, including Tunnel and operations on tables and projects.
<code>odps-sdk-commons</code>	Provides util encapsulation.
<code>odps-sdk-udf</code>	Provides main interfaces of UDFs.
<code>odps-sdk-mapred</code>	Provides the MapReduce function.
<code>odps-sdk-graph</code>	Provides Graph Java SDK. You can search by the keyword <code>odps-sdk-graph</code> .

UDAF

Java UDAFs must inherit `com.aliyun.odps.udf.Aggregator` and implement the following interfaces:

```

import com.aliyun.odps.udf.ContextFunction;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDFException;
public abstract class Aggregator implements ContextFunction {
    @Override
    public void setup(ExecutionContext ctx) throws UDFException {
    }
    @Override
    public void close() throws UDFException {
    }
    /**
     * Create an aggregation buffer * @return Writable aggregation buffer
     */
    abstract public Writable newBuffer();
    /**
     * @param buffer: aggregation buffer * @param args: a parameter specified to call a UDAF in SQL. It
     cannot be NULL, but the values in args can be NULL, which indicates that the input data is NULL * @thr
     ows UDFException.
     */
    abstract public void iterate(Writable buffer, Writable[] args) throws UDFException;
    /**
     * Generate the final result * @param buffer * @return Final result of Object UDAF * @throws UDFExc
     eption.
     */
    abstract public Writable terminate(Writable buffer) throws UDFException;
    abstract public void merge(Writable buffer, Writable partial) throws UDFException;
}

```

The most important interfaces are `iterate`, `merge`, and `terminate` because the main logic of UDAFs relies on these interfaces. In addition, you must implement the user-defined Writable buffer.

The following figure illustrates the implementation logic and calculation procedure of the `avg` function in MaxCompute UDAFs.

□

In the preceding figure, the input data is sliced according to the specified size. (For more information, see [MapReduce](#)). The size of each slice is suitable for a worker to complete in a specified time. The slice size must be manually configured.

The calculation procedure of a UDAF involves two steps:

- Step 1: Each worker counts the data quantity and total sum in a slice. You can consider the data quantity and total sum in each slice as an intermediate result.

- Step 2: Each worker gathers the information about each slice generated in step 1. In the final output, `r.sum/r.count` is the average value of all input data.

The following code sample implements a UDAF that calculates the average of numbers:

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.udf.Aggregator;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.annotation.Resolve;
@Resolve("double->double")
public class AggrAvg extends Aggregator {
    private static class AvgBuffer implements Writable {
        private double sum = 0;
        private long count = 0;
        @Override
        public void write(DataOutput out) throws IOException {
            out.writeDouble(sum);
            out.writeLong(count);
        }
        @Override
        public void readFields(DataInput in) throws IOException {
            sum = in.readDouble();
            count = in.readLong();
        }
    }
    private DoubleWritable ret = new DoubleWritable();
    @Override
    public Writable newBuffer() {
        return new AvgBuffer();
    }
    @Override
    public void iterate(Writable buffer, Writable[] args) throws UDFException {
        DoubleWritable arg = (DoubleWritable) args[0];
        AvgBuffer buf = (AvgBuffer) buffer;
        if (arg != null) {
            buf.count += 1;
            buf.sum += arg.get();
        }
    }
}
```

```

}
@Override
public Writable terminate(Writable buffer) throws UDFException {
    AvgBuffer buf = (AvgBuffer) buffer;
    if (buf.count == 0) {
        ret.set(0);
    } else {
        ret.set(buf.sum / buf.count);
    }
    return ret;
}
@Override
public void merge(Writable buffer, Writable partial) throws UDFException {
    AvgBuffer buf = (AvgBuffer) buffer;
    AvgBuffer p = (AvgBuffer) partial;
    buf.sum += p.sum;
    buf.count += p.count;
}
}

```

The following code sample implements Concat by using a Writable type:

```

package com.aliyun.odps.udf.example;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.udf.UDF;
public class MyConcat extends UDF {
    private Text ret = new Text();
    public Text evaluate(Text a, Text b) {
        if (a == null || b == null) {
            return null;
        }
        ret.clear();
        ret.append(a.getBytes(), 0, a.getLength());
        ret.append(b.getBytes(), 0, b.getLength());
        return ret;
    }
}

```

- `Writable[] writables` : indicates a row of data. In the code, it indicates the passed column. For example, `writables[0]` indicates the first column, and `writables[1]` indicates the second column.

- `iterate(Writable writable,Writable[] writables)` method: `writable` indicates the aggregated data in a phase. In different Map tasks, each row of the data (also regarded as a set) obtained by using `group by` is executed once.
- `merge()` method: aggregates the results obtained from different Map tasks.
- `terminate()` method: returns data.
- `newBuffer()` method: creates the initial value to be returned.
- In the `readFields` method of `Writable`, the `readFields` method of the same object is called multiple times because the `writable` objects of partial are reused. This method resets the entire object each time it is called. If an object contains `Collection`, it is cleared.
- UDAFs are similar to aggregate functions in MaxCompute SQL. For more information, see [Aggregate functions](#).
- The method for running UDTFs is similar to that for running UDFs. For more information, see [Develop Java UDFs \(Optional\)](#).
- The `Writable` type that corresponds to `STRING` is `Text`.

UDTF

Java UDTFs must inherit the `com.aliyun.odps.udf.UDTF` class and implement four interfaces. The following table describes the interfaces.

Interface definition	Description
<code>public void setup(ExecutionContext ctx) throws UDFException</code>	The initialization method to call the user-defined initialization behavior before a UDTF processes the input data. <code>setup</code> is called once first for each worker.
<code>public void process(Object[] args) throws UDFException</code>	This method is called by the framework. Each SQL record calls <code>process</code> once. The parameters of <code>process</code> are the input parameters of the UDTF specified in the SQL statement. The input parameters are passed in as <code>Object[]</code> , and the results are returned by using the <code>forward</code> function. You must call <code>forward</code> in the <code>process</code> function to determine the output data.
<code>public void close() throws UDFException</code>	The method for terminating a UDTF. The framework calls this method only once. The method is called after the last record is processed.
<code>public void forward(Object ...o) throws UDFException</code>	You can call the <code>forward</code> method to return data. One record is returned each time <code>forward</code> is called. The record corresponds to the column specified by the <code>as</code> clause in the SQL statement of the UDTF.

Example

1. The following code sample shows a UDTF.


```

package org.alidata.odps.udtf.examples;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.UDTFCollector;
import com.aliyun.odps.udf.annotation.Resolve;
import com.aliyun.odps.udf.UDFException;
// TODO define input and output types, e.g., "string,string->string,bigint".
@Resolve("string,bigint->string,bigint")
public class MyUDTF extends UDTF {
    @Override
    public void process(Object[] args) throws UDFException {
        String a = (String) args[0];
        Long b = (Long) args[1];
        for (String t: a.split("\\s+")) {
            forward(t, b);
        }
    }
}

```

 **Note** The method for running UDTFs in MaxCompute is similar to that for running UDFs. For more information, see [Develop Java UDFs \(Optional\)](#).

2. Assume that you create a UDTF in MaxCompute with the registered function name of `user_udtf`. Execute the following statement to use the UDTF:

```
select user_udtf(col0, col1) as (c0, c1) from my_table;
```

The following example shows the values of `col0` and `col1` in `my_table`:

```

+-----+-----+
| col0 | col1 |
+-----+-----+
| A B | 1 |
| C D | 2 |
+-----+-----+

```

The following example shows the result of `select`:

```
+----+----+
| c0 | c1 |
+----+----+
| A | 1 |
| B | 1 |
| C | 2 |
| D | 2 |
+----+----+
```

Example of using UDTFs to read resources in MaxCompute

You can use UDTFs to read resources in MaxCompute. For more information, see [Resource](#). The following example describes how to use a UDTF to read resources in MaxCompute:

1. Compile a UDTF program. After the compilation is successful, export the JAR package. In this example, the JAR package is named *udtfexample1.jar*.

```
package com.aliyun.odps.examples.udf;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Iterator;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.annotation.Resolve;

/**
 * project: example_project
 * table: wc_in2
 * partitions: p2=1,p1=2
 * columns: colc,colb
 */
@Resolve("string,string->string,bigint,string")
public class UDTFResource extends UDTF {
    ExecutionContext ctx;
    long fileResourceLineCount;
    long tableResource1RecordCount;
    long tableResource2RecordCount;
    @Override
    public void setup(ExecutionContext ctx) throws UDFException {
        this.ctx = ctx;
        try {
```

```

    InputStream in = ctx.readResourceFileAsStream("file_resource.txt");
    BufferedReader br = new BufferedReader(new InputStreamReader(in));
    String line;
    fileResourceLineCount = 0;
    while ((line = br.readLine()) != null) {
        fileResourceLineCount++;
    }
    br.close();
    Iterator<Object[]> iterator = ctx.readResourceTable("table_resource1").iterator();
    tableResource1RecordCount = 0;
    while (iterator.hasNext()) {
        tableResource1RecordCount++;
        iterator.next();
    }
    iterator = ctx.readResourceTable("table_resource2").iterator();
    tableResource2RecordCount = 0;
    while (iterator.hasNext()) {
        tableResource2RecordCount++;
        iterator.next();
    }
} catch (IOException e) {
    throw new UDFException(e);
}
}

@Override
public void process(Object[] args) throws UDFException {
    String a = (String) args[0];
    long b = args[1] == null ? 0 : ((String) args[1]).length();
    forward(a, b, "fileResourceLineCount=" + fileResourceLineCount + "|tableResource1RecordCount="
+ tableResource1RecordCount + "|tableResource2RecordCount=" + tableResource2RecordCount
);
}
}

```

2. Add resources to MaxCompute.

```

Add file file_resource.txt;
Add jar udtfexample1.jar;
Add table table_resource1 as table_resource1;
Add table table_resource2 as table_resource2;

```

3. Create a UDTF in MaxCompute. In this example, the UDTF is named `mp_udtf`.

```
create function mp_udtf as com.aliyun.odps.examples.udf.UDTFResource using
'udtfexample1.jar, file_resource.txt, table_resource1, table_resource2';
```

4. Run this UDTF.

```
select mp_udtf("10","20") as (a, b, fileResourceLineCount) from tmp1;
-- Returned result:
+-----+-----+-----+
| a | b | fileResourceLineCount |
+-----+-----+-----+
| 10 | 2 | fileResourceLineCount=3|tableResource1RecordCount=0|tableResource2RecordCou
nt=0 |
| 10 | 2 | fileResourceLineCount=3|tableResource1RecordCount=0|tableResource2RecordCou
nt=0 |
+-----+-----+-----+
```

Examples of using complex data types in a UDF

The following example defines a UDF with three overloads. The first overload uses the `ARRAY` type, the second uses the `MAP` type, and the third uses the `STRUCT` type. The third overload uses `STRUCT` as the data type of parameters or returned values. Therefore, the `@Resolve` annotation must be added to the UDF class to specify the `STRUCT` type.

```
import com.aliyun.odps.udf.UDF;
import com.aliyun.odps.udf.annotation.Resolve;
@Resolve("struct,string->string")
public class UdfArray extends UDF {
    public String evaluate(List vals, Long len) {
        return vals.get(len.intValue());
    }

    public String evaluate(Map map, String key) {
        return map.get(key);
    }

    public String evaluate(Struct struct, String key) {
        return struct.getFieldValue("a") + key;
    }
}
```

You can import complex data types into a UDF.

```
create function my_index as 'UdfArray' using 'myjar.jar';  
select id, my_index(array('red', 'yellow', 'green'), colorOrdinal) as color_name from co
```

Hive UDF compatibility example

MaxCompute V2.0 supports Hive UDFs. Some Hive UDFs can be directly used in MaxCompute.

 **Note** MaxCompute V2.0 is compatible with Hive 2.1.0, and the corresponding Hadoop version is 2.7.2. If you develop a UDF by using other versions of Hive or Hadoop, you may need to recompile the UDF on a compatible version.

The following example shows how to use a Hive UDF in MaxCompute:

```
package com.aliyun.odps.compiler.hive;
import org.apache.hadoop.hive ql.exec.UDFArgumentException;
import org.apache.hadoop.hive ql.metadata.HiveException;
import org.apache.hadoop.hive ql.udf.generic.GenericUDF;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;
import org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
public class Collect extends GenericUDF {
    @Override
    public ObjectInspector initialize(ObjectInspector[] objectInspectors) throws UDFArgumentException {
        if (objectInspectors.length == 0) {
            throw new UDFArgumentException("Collect: input args should >= 1");
        }
        for (int i = 1; i < objectInspectors.length; i++) {
            if (objectInspectors[i] != objectInspectors[0]) {
                throw new UDFArgumentException("Collect: input oi should be the same for all args");
            }
        }
        return ObjectInspectorFactory.getStandardListObjectInspector(objectInspectors[0]);
    }
    @Override
    public Object evaluate(DeferredObject[] deferredObjects) throws HiveException {
        List<Object> objectList = new ArrayList<>(deferredObjects.length);
        for (DeferredObject deferredObject : deferredObjects) {
            objectList.add(deferredObject.get());
        }
        return objectList;
    }
    @Override
    public String getDisplayString(String[] strings) {
        return "Collect";
    }
}
```

 Note

- The UDF in the example supports all data types, such as ARRAY, MAP, and STRUCT.
- For more information about how to use Hive UDFs, see the following references:
 - [HivePlugins](#)
 - [DeveloperGuide UDTF](#)
 - [GenericUDAFCaseStudy](#)

The UDF in the example packages any type and number of parameters into ARRAY. Assume that the output jar package is named `test.jar`.

```
-- Add a resource.
Add jar test.jar;
-- Create a function.
CREATE FUNCTION hive_collect as 'com.aliyun.odps.compiler.hive.Collect' using 'test.jar';
-- Use the function.
set odps.sql.hive.compatible=true;
select hive_collect(4y,5y,6y) from dual;
+-----+
| _c0 |
+-----+
| [4, 5, 6] |
+-----+
```

Consider the following notes when you use Hive-compatible UDFs in MaxCompute:

- Specify the JAR package when you create a UDF. MaxCompute does not automatically add all JAR packages to the classpath.

 **Note** The `add jar` command in MaxCompute creates a permanent resource in the project.

- Insert `set odps.sql.hive.compatible=true;` before an SQL statement, and commit and execute the `set` statement with the SQL statement.
- Pay attention to the limits imposed by the [Java sandbox](#) on MaxCompute.

3.16.3. Python 2 UDFs

MaxCompute supports three types of Python 2 user-defined functions (UDFs): UDFs, user-defined aggregate functions (UDAFs), and user-defined table-valued functions (UDTFs). This topic describes how to implement these Python 2 UDFs.

Limits

MaxCompute uses Python 2.7. User code is run in a sandbox, which is a restricted environment. In this environment, the following operations are prohibited:

- Read and write local files.
- Start subprocesses.
- Start threads.
- Initiate socket-based communication.
- Call other systems.

Due to these limits, all code that you upload must be implemented by standard Python, and C extension modules are not allowed in your code.

In addition, all modules that involve the preceding operations are disabled in the Python standard library. The following modules are available in the standard library:

- All modules that are implemented by standard Python.
- The following C extension modules:
 - array and audioop
 - binascii and bisect
 - cmath, _codecs_cn, _codecs_hk, _codecs_iso2022, _codecs_jp, _codecs_kr, _codecs_tw, _collections, and cStringIO
 - datetime
 - _functools and future_builtins
 - _heapq and _hashlib
 - itertools
 - _json
 - _locale and _lsprof
 - math, _md5, and _multibytecodec
 - operator
 - _random
 - _sha256, _sha512, _sha, _struct, and strop
 - time
 - unicodedata
 - _weakref
 - cPickle
- Some modules with limited functionality. For example, a sandbox limits the size of data that your code can write to the standard output and standard error output, namely, `sys.stdout` and `sys.stderr`, to 20 KB. Characters that exceed this limit are ignored.

Third-party libraries

Common third-party libraries such as NumPy are installed in the runtime environment of Python 2 to supplement the standard library.

 **Note** The use of third-party libraries is also subject to limits. For example, when you use a third-party library, you are not allowed to access local data and you can use only limited network I/O resources. The related APIs in the third-party libraries are disabled.

Data types of parameters and return values

The data types of parameters and return values are specified by the following method:

```
@odps.udf.annotate(signature)
```

Python UDFs support the following MaxCompute SQL data types: BIGINT, STRING, DOUBLE, BOOLEAN, DATETIME, DECIMAL, ARRAY, MAP, STRUCT, and nested complex data types. Complex data types include ARRAY, MAP, and STRUCT. Before you execute an SQL statement, the data types of parameters and return values of all functions must be determined. Python is a dynamically typed language. You must add decorators to the UDF class to specify the function signature.

The function signature is specified by a string. The following syntax is supported:

```
arg_type_list '->' type_list
arg_type_list: type_list | '*' | " | 'char(n)' | 'varchar(n)'
type_list: [type_list ','] type
type: 'bigint' | 'string' | 'double' | 'boolean' | 'datetime' | 'float' | 'binary' | 'date' | 'decimal' | 'decimal(precision,scale)'
```

- The part to the left of the arrow indicates the data types of parameters. The part to the right of the arrow indicates the data types of return values.
- The return value of a UDTF can contain multiple columns, whereas that of a UDF or UDAF contains only one column.
- An asterisk (*) represents variable-length parameters. If variable-length parameters are used, UDFs, UDTFs, and UDAFs can match input parameters of any data type.
- In a project that uses the MaxCompute V2.0 data type edition, you can set the decimal parameter to precision or scale. For more information, see [MaxCompute V2.0 data type edition](#).
- You cannot specify the CHAR or VARCHAR type for return values.

The following example shows a valid signature:

```
'bigint,double->string'      # The parameters are of the BIGINT and DOUBLE types and the return values are of the STRING type.
'bigint,boolean->string,datetime' # The parameters are of the BIGINT and BOOLEAN types and the return values are of the STRING and DATETIME types.
'*->string'                  # The parameters are variable-length parameters and can be of any data type, and the return values are of the STRING type.
'->double'                  # The parameters are left empty and the return values are of the DOUBLE type.
.
'array<bigint>->struct<x:string, y:int>' # The parameters are of the ARRAY<BIGINT> type and the return values are of the STRUCT<x:STRING, y:INT> type.
'->map<bigint, string>'      # The parameters are left empty and the return values are of the MAP<BIGINT, STRING> type.
```

If an invalid signature is found during query parsing, an error is returned and the execution of the function is prohibited. During the execution of the function, the parameters of the type that is specified by the function signature are transferred. The return values must be of the same type as that specified by the function signature. Otherwise, an error is returned.

The following table lists the mappings between MaxCompute SQL data types and Python 2 data types.

MaxCompute SQL data type	Python 2 data type
BIGINT	INT
STRING	STR
DOUBLE	FLOAT
BOOLEAN	BOOL
DATETIME	INT
FLOAT	FLOAT
CHAR	STR
VARCHAR	STR
BINARY	BYTEARRAY
DATE	INT
DECIMAL	DECIMAL.DECIMAL
ARRAY	LIST
MAP	DICT

MaxCompute SQL data type	Python 2 data type
STRUCT	COLLECTIONS.NAMEDTUPLE

? Note

- A value of the DATETIME type is converted to a value of the INT type. The value of the INT type indicates the number of milliseconds that have elapsed since the epoch time January 1, 1970, 00:00:00 UTC. You can process data of the DATETIME type by using the datetime module in the Python standard library.
- `silent` is added to `odps.udf.int(value,[silent=True])`. If `silent` is set to True and `value` cannot be converted to the INT type, None is returned, and no error is reported.
- NULL in MaxCompute SQL corresponds to None in Python.

UDFs

To implement Python UDFs, you must define a new-style class and implement the `evaluate` method.

```
from odps.udf import annotate
@annotate("bigint,bigint->bigint")
class MyPlus(object):
    def evaluate(self, arg0, arg1):
        if None in (arg0, arg1):
            return None
        return arg0 + arg1
```

? Note A Python UDF must have its signature specified by using `annotate`.

UDAFs

- `class odps.udf.BaseUDAF` : the base class of Python UDAFs. It can be inherited to implement Python UDAFs.
- `BaseUDAF.new_buffer()` : returns `buffer` of the intermediate value of a UDAF. `buffer` must be a **marshallable** object such as a list or a dictionary, and the `buffer` size cannot increase with the data volume. Under extreme circumstances, the `buffer` size cannot exceed 2 MB after the marshal operation.
- `BaseUDAF.iterate(buffer[, args, ...])` : aggregates `args` to `buffer` of the intermediate value.
- `BaseUDAF.merge(buffer, pBuffer)` : aggregates `buffer` of two intermediate values. It merges `pBuffer` to `buffer`.
- `BaseUDAF.terminate(buffer)` : converts `buffer` of the intermediate value to a value of a basic data type in MaxCompute SQL.

The following example shows how to use a UDAF to obtain the average value:

```
@annotate('double->double')
class Average(BaseUDAF):
    def new_buffer(self):
        return [0, 0]
    def iterate(self, buffer, number):
        if number is not None:
            buffer[0] += number
            buffer[1] += 1
    def merge(self, buffer, pBuffer):
        buffer[0] += pBuffer[0]
        buffer[1] += pBuffer[1]
    def terminate(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
```

UDTFs

- `class odps.udf.BaseUDTF` : the base class of Python UDTFs. It can be inherited to implement the `process` and `close` methods.
- `BaseUDTF.__init__()` : the initialization method. To implement this method for a derived class, you must call the `super(BaseUDTF, self).__init__()` initialization method of the base class at the beginning of code execution. Throughout the lifecycle of a UDTF, the `init` method is called only once. It is called only before the first record is processed. If a UDTF needs to save internal states, you can call the `init` method to initialize all states.
- `BaseUDTF.process([args, ...])` : is called by the MaxCompute SQL framework. The `process` method is called to process each record involved in the SQL statements. The UDTF input parameters that are specified in the SQL statements are used as the parameters of the `process` method.
- `BaseUDTF.forward([args, ...])` : the UDTF output method. This method is called by user code. One record is generated each time the `forward` method is called. The UDTF output parameters that are specified in the SQL statements are used as the parameters of the `forward` method.
- `BaseUDTF.close()` : the UDTF termination method. Throughout the lifecycle of a UDTF, this method is called only once by the MaxCompute SQL framework. It is called only after the last record is processed.

The following example shows how to use a UDTF:

```
#coding:utf-8
# explode.py
from odps.udf import annotate
from odps.udf import BaseUDTF
@annotate('string -> string')
class Explode(BaseUDTF):
    """ Use commas (,) to separate the string into multiple records.
    def process(self, arg):
        props = arg.split(',')
        for p in props:
            self.forward(p)
```

 **Note** The data types of parameters and return values of a Python UDTF can be specified without the need to use `annotate` to specify the UDTF signature. In this case, the UDTF can match input parameters of any data type in SQL statements. However, the data types of return values cannot be inferred. All output parameters are considered to be of the `STRING` type. Therefore, when the `forward` method is called, all output values must be converted to the `STRING` type.

Resource reference

You can reference file and table resources in Python UDFs by using the `odps.distcache` module.

- `odps.distcache.get_cache_file(resource_name)` : returns the content of a specific file resource.
 - `resource_name` is a string that corresponds to the name of an existing file resource in the current project. If the resource name is invalid or the file resource does not exist, an error is returned.

 **Note** To reference a file resource in a UDF, you must declare the file resource when you create the UDF. Otherwise, an error is returned when the UDF is called.

- The return value is a file-like object. If this object is no longer used, the caller must call the `close` method to release the open file resource.

Example:

```
@annotate('bigint->string')
class DistCacheExample(object):
def __init__(self):
    cache_file = get_cache_file('test_distcache.txt')
    kv = {}
    for line in cache_file:
        line = line.strip()
        if not line:
            continue
        k, v = line.split()
        kv[int(k)] = v
    cache_file.close()
    self.kv = kv
def evaluate(self, arg):
    return self.kv.get(arg)
```

- `odps.distcache.get_cache_table(resource_name)` : returns the content of a specific table resource.
 - `resource_name` is a string that corresponds to the name of an existing table resource in the current project. If the resource name is invalid or the table resource does not exist, an error is returned.
 - The return value is of the generator type. The caller traverses the table to obtain the content. Each time the caller traverses the table, a record of the ARRAY type is obtained.

Example:

```
from odps.udf import annotate
from odps.distcache import get_cache_table
@annotate('->string')
class DistCacheTableExample(object):
def __init__(self):
    self.records = list(get_cache_table('udf_test'))
    self.counter = 0
    self.ln = len(self.records)
def evaluate(self):
    if self.counter > self.ln - 1:
        return None
    ret = self.records[self.counter]
    self.counter += 1
    return str(ret)
```

3.16.4. Python 3 UDFs

Python Software Foundation announces the End of Life (EOL) for Python 2. Due to this reason, MaxCompute supports Python 3 and uses CPython 3.7.3. MaxCompute supports three types of Python 3 user-defined functions (UDFs): UDFs, user-defined aggregate functions (UDAFs), and user-defined table-valued functions (UDTFs). This topic describes how to implement these Python 3 UDFs.

Limits

Python 3 is incompatible with Python 2. You can select Python 3 or Python 2 to execute an SQL statement, but cannot use both Python 3 and Python 2 code in a single SQL statement.

Enable Python 3

You can enable Python 3 only for jobs. To enable Python 3 for a job, insert the following statement before the corresponding SQL statement and execute them together:

```
set odps.sql.python.version=cp37;
```

Third-party libraries

MaxCompute Python UDFs support third-party libraries. In addition to the standard library, the third-party library NumPy is installed in the Python 2 runtime environment to supplement the standard library.

NumPy is not installed in the Python 3 runtime environment in MaxCompute. To use a NumPy UDF, you must manually upload a NumPy WHL file. If you download a NumPy WHL file from Python Package Index (PyPI) or an image, the file name is *numpy-<Version>-cp37-cp37m-manylinux1_x86_64.whl*. For more information about how to upload a file, see [Resource operations](#).

Port Python 2 UDFs

Python Software Foundation announces the EOL for Python 2. Therefore, we recommend that you port Python 2 UDFs. The method of porting Python 2 UDFs varies based on the project.

- In a new project or an existing project where no Python UDFs exist, we recommend that you use Python 3 to write all Python UDFs.
- In an existing project where a large number of Python 2 UDFs exist, exercise caution when you enable Python 3. You cannot use both Python 2 and Python 3 code in a single SQL statement. If you plan to gradually replace Python 2 UDFs with Python 3 UDFs, use the following methods:
 - Use Python 3 to write new UDFs and enable Python 3 for new jobs.
 - Rewrite Python 2 UDFs to make them compatible with both Python 2 and Python 3. For more information about how to rewrite UDFs, see [Porting Python 2 Code to Python 3](#).

 **Note** When you write a public UDF that needs to be shared among multiple projects, we recommend that you write the UDF to be compatible with both Python 2 and Python 3.

Data types of parameters and return values

The data types of parameters and return values are specified by the following method:

```
@odps.udf.annotate(signature)
```

Python UDFs support the following MaxCompute SQL data types: BIGINT, STRING, DOUBLE, BOOLEAN, DATETIME, DECIMAL, ARRAY, MAP, STRUCT, and nested complex data types. Complex data types include ARRAY, MAP, and STRUCT. Before you execute an SQL statement, the data types of parameters and return values of all functions must be determined. Python is a dynamically typed language. You must add decorators to the UDF class to specify the function signature.

The function signature is specified by a string. The following syntax is supported:

```
arg_type_list '->' type_list
arg_type_list: type_list | '*' | " | 'char(n)' | 'varchar(n)'
type_list: [type_list ','] type
type: 'bigint' | 'string' | 'double' | 'boolean' | 'datetime' | 'float' | 'binary' | 'date' | 'decimal' | 'decimal(precision,scale)'
```

- The part to the left of the arrow indicates the data types of parameters. The part to the right of the arrow indicates the data types of return values.
- The return value of a UDTF can contain multiple columns, whereas that of a UDF or UDAF contains only one column.
- An asterisk (*) represents variable-length parameters. If variable-length parameters are used, UDFs, UDTFs, and UDAFs can match input parameters of any data type.
- In a project that uses the MaxCompute V2.0 data type edition, you can set the decimal parameter to precision or scale. For more information, see [MaxCompute V2.0 data type edition](#).
- You cannot specify the CHAR or VARCHAR type for return values.

The following example shows a valid signature:

```
'bigint,double->string'      # The parameters are of the BIGINT and DOUBLE types and the return values are of the STRING type.
'bigint,boolean->string,datetime' # The parameters are of the BIGINT and BOOLEAN types and the return values are of the STRING and DATETIME types.
'*->string'                  # The parameters are variable-length parameters and can be of any data type, and the return values are of the STRING type.
'->double'                  # The parameters are left empty and the return values are of the DOUBLE type.
.
'array<bigint>->struct<x:string, y:int>' # The parameters are of the ARRAY<BIGINT> type and the return values are of the STRUCT<x:STRING, y:INT> type.
'->map<bigint, string>'      # The parameters are left empty and the return values are of the MAP<BIGINT, STRING> type.
```


If an invalid signature is found during query parsing, an error is returned and the execution of the function is prohibited. During the execution of the function, the parameters of the type that is specified by the function signature are transferred. The return values must be of the same type as that specified by the function signature. Otherwise, an error is returned.

The following table lists the mappings between MaxCompute SQL data types and Python 3 data types.

MaxCompute SQL data type	Python 3 data type
BIGINT	INT
STRING	UNICODE
DOUBLE	FLOAT
BOOLEAN	BOOL
DATETIME	DATETIME.DATETIME
FLOAT	FLOAT
CHAR	UNICODE
VARCHAR	UNICODE
BINARY	BYTES
DATE	DATETIME.DATE
DECIMAL	DECIMAL.DECIMAL
ARRAY	LIST
MAP	DICT
STRUCT	COLLECTIONS.NAMEDTUPLE

Note

- A value of the DATETIME type is converted to a value of the INT type. The value of the INT type indicates the number of milliseconds that have elapsed since the epoch time January 1, 1970, 00:00:00 UTC. You can process data of the DATETIME type by using the datetime module in the Python standard library.
- `silent` is added to `odps.udf.int(value,[silent=True])`. If `silent` is set to True and `value` cannot be converted to the INT type, None is returned, and no error is reported.
- NULL in MaxCompute SQL corresponds to None in Python.

UDFs

To implement Python UDFs, you must define a new-style class and implement the `evaluate` method.

```
from odps.udf import annotate
@annotate("bigint,bigint->bigint")
class MyPlus(object):
    def evaluate(self, arg0, arg1):
        if None in (arg0, arg1):
            return None
        return arg0 + arg1
```

 **Note** A Python UDF must have its signature specified by using `annotate` .

UDAFs

- `class odps.udf.BaseUDAF` : the base class of Python UDAFs. It can be inherited to implement Python UDAFs.
- `BaseUDAF.new_buffer()` : returns `buffer` of the intermediate value of a UDAF. `buffer` must be a **marshallable** object such as a list or a dictionary, and the `buffer` size cannot increase with the data volume. Under extreme circumstances, the `buffer` size cannot exceed 2 MB after the marshal operation.
- `BaseUDAF.iterate(buffer[, args, ...])` : aggregates `args` to `buffer` of the intermediate value.
- `BaseUDAF.merge(buffer, pBuffer)` : aggregates `buffer` of two intermediate values. It merges `pBuffer` to `buffer` .
- `BaseUDAF.terminate(buffer)` : converts `buffer` of the intermediate value to a value of a basic data type in MaxCompute SQL.

The following example shows how to use a UDAF to obtain the average value:

```

@annotate('double->double')
class Average(BaseUDAF):
    def new_buffer(self):
        return [0, 0]
    def iterate(self, buffer, number):
        if number is not None:
            buffer[0] += number
            buffer[1] += 1
    def merge(self, buffer, pBuffer):
        buffer[0] += pBuffer[0]
        buffer[1] += pBuffer[1]
    def terminate(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]

```

UDTFs

- `class odps.udf.BaseUDTF` : the base class of Python UDTFs. It can be inherited to implement the `process` and `close` methods.
- `BaseUDTF.__init__()` : the initialization method. To implement this method for a derived class, you must call the `super(BaseUDTF, self).__init__()` initialization method of the base class at the beginning of code execution. Throughout the lifecycle of a UDTF, the `init` method is called only once. It is called only before the first record is processed. If a UDTF needs to save internal states, you can call the `init` method to initialize all states.
- `BaseUDTF.process([args, ...])` : is called by the MaxCompute SQL framework. The `process` method is called to process each record involved in the SQL statements. The UDTF input parameters that are specified in the SQL statements are used as the parameters of the `process` method.
- `BaseUDTF.forward([args, ...])` : the UDTF output method. This method is called by user code. One record is generated each time the `forward` method is called. The UDTF output parameters that are specified in the SQL statements are used as the parameters of the `forward` method.
- `BaseUDTF.close()` : the UDTF termination method. Throughout the lifecycle of a UDTF, this method is called only once by the MaxCompute SQL framework. It is called only after the last record is processed.

The following example shows how to use a UDTF:

```
#coding:utf-8
# explode.py
from odps.udf import annotate
from odps.udf import BaseUDTF
@annotate('string -> string')
class Explode(BaseUDTF):
    """ Use commas (,) to separate the string into multiple records.
    def process(self, arg):
        props = arg.split(',')
        for p in props:
            self.forward(p)
```

 **Note** The data types of parameters and return values of a Python UDTF can be specified without the need to use `annotate` to specify the UDTF signature. In this case, the UDTF can match input parameters of any data type in SQL statements. However, the data types of return values cannot be inferred. All output parameters are considered to be of the `STRING` type. Therefore, when the `forward` method is called, all output values must be converted to the `STRING` type.

Resource reference

You can reference file and table resources in Python UDFs by using the `odps.distcache` module.

- `odps.distcache.get_cache_file(resource_name)` : returns the content of a specific file resource.
 - `resource_name` is a string that corresponds to the name of an existing file resource in the current project. If the resource name is invalid or the file resource does not exist, an error is returned.

 **Note** To reference a file resource in a UDF, you must declare the file resource when you create the UDF. Otherwise, an error is returned when the UDF is called.

- The return value is a file-like object. If this object is no longer used, the caller must call the `close` method to release the open file resource.

Example:

```

@annotate('bigint->string')
class DistCacheExample(object):
    def __init__(self):
        cache_file = get_cache_file('test_distcache.txt')
        kv = {}
        for line in cache_file:
            line = line.strip()
            if not line:
                continue
            k, v = line.split()
            kv[int(k)] = v
        cache_file.close()
        self.kv = kv
    def evaluate(self, arg):
        return self.kv.get(arg)

```

- `odps.distcache.get_cache_table(resource_name)` : returns the content of a specific table resource.
 - `resource_name` can be of the BIGINT, STRING, DOUBLE, BOOLEAN, DATETIME, FLOAT, CHAR, VARCHAR, BINARY, DATE, DECIMAL, ARRAY, MAP, or STRUCT type. It corresponds to the name of an existing table resource in the current project. If the resource name is invalid or the table resource does not exist, an error is returned.
 - The return value is of the generator type. The caller traverses the table to obtain the content. Each time the caller traverses the table, a record of the ARRAY type is obtained.

Example:

```

from odps.udf import annotate
from odps.distcache import get_cache_table
@annotate('->string')
class DistCacheTableExample(object):
    def __init__(self):
        self.records = list(get_cache_table('udf_test'))
        self.counter = 0
        self.ln = len(self.records)
    def evaluate(self):
        if self.counter > self.ln - 1:
            return None
        ret = self.records[self.counter]
        self.counter += 1
        return str(ret)

```

3.16.5. Code-embedded UDFs

This topic describes how to use code-embedded user-defined functions (UDFs) to embed Java or Python code into SQL scripts.

Background information

Code-embedded UDFs of MaxCompute resolve the following issues in code implementation and maintenance:

- **Complicated code implementation:** After you create UDFs and develop code, you must compile the code in Java and create resources and functions.
- **Inconvenient code maintenance:** You cannot directly view the implementation logic of UDFs that are referenced in SQL scripts or obtain the source code of JAR packages.
- **Poor code readability:** To implement Java library functions by using user-defined type (UDT), you must write Java code as expressions in long code lines. In addition, you may fail to write some Java code as expressions. Example:

```
Foo f = new Foo();  
f.execute();  
f.getResult();
```

Features

Code-embedded UDFs allow you to embed Java or Python code into SQL scripts. When you compile a script, Janino-compiler identifies and extracts the embedded code, compiles the code in Java, and then dynamically generates resources and creates temporary functions.

You can place SQL scripts and third-party code lines in the same source code file. This simplifies the usage of UDT or UDFs and facilitates daily development and maintenance.

Limits

You can use only Janino-compiler to compile embedded Java code, and the syntax of the embedded Java code must be a subset of the standard JDK syntax. Embedded Java code has the following limits:

- It does not support lambda expressions.
- You cannot specify multiple types of exceptions in a single catch block. For example, `catch(Exception1 | Exception2 e)` is not allowed.
- It cannot automatically infer generic arguments. For example, `Map map = new HashMap<>();` is not supported.
- Expressions for type argument inference are ignored. You must use cast expressions to specify the argument type, for example, `(String) myMap.get(key)`.
- Assertions are forcibly enabled, even if the `-ea` option of the Java virtual machine (JVM) is used.
- Code that is programmed in Java 8 or later versions is not supported.

Reference embedded code in UDT

You must submit and execute the script in Script Mode SQL. For more information, see [Script Mode SQL](#). Sample code:

```
SELECT
  s,
  com.mypackage.Foo.extractNumber(s)
FROM VALUES ('abc123def'),('apple') AS t(s);

#CODE ('lang'='JAVA')
package com.mypackage;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
public class Foo {
  final static Pattern compile = Pattern.compile(".*?([0-9]+).*");
  public static String extractNumber(String input) {
    final Matcher m = compile.matcher(input);
    if (m.find()) {
      return m.group(1);
    }
    return null;
  }
}
#END CODE;
```

- `#CODE` and `#END CODE` indicate the beginning and end of the embedded code block, respectively. In this example, the embedded code block is placed at the end of the script, and applies to the whole script.
- `'lang'='JAVA'` indicates that the embedded code is in Java. JAVA can be replaced with `PYTHON`.
- You can use UDT syntax in the SQL script to call `Foo.extractNumber`.

Define and call a Java code-embedded UDF

Sample code:

```
CREATE TEMPORARY FUNCTION foo AS 'com.mypackage.Reverse' USING
#CODE ('lang'='JAVA')
package com.mypackage;
import com.aliyun.odps.udf.UDF;
public class Reverse extends UDF {
    public String evaluate(String input) {
        if (input == null) return null;
        StringBuilder ret = new StringBuilder();
        for (int i = input.toCharArray().length - 1; i >= 0; i--) {
            ret.append(input.toCharArray()[i]);
        }
        return ret.toString();
    }
}
#END CODE;

SELECT foo('abdc');
```

- You can place the embedded code block next to `USING` or at the end of the script. If you place it next to `USING`, it applies only to the `CREATE TEMPORARY FUNCTION` statement.
- The function created by `CREATE TEMPORARY FUNCTION` is a temporary function. This temporary function is executed only during the current execution process and is not stored in the MaxCompute meta system.

Define and call a Java code-embedded UDTF

Sample code:


```

CREATE TEMPORARY FUNCTION foo AS 'com.mypackage.Reverse' USING
#CODE ('lang'='JAVA', 'filename'='embedded.jar')
package com.mypackage;

import com.aliyun.odps.udf.UDTF;
import com.aliyun.odps.udf.UDFException;
import com.aliyun.odps.udf.annotation.Resolve;

@Resolve({"string->string,string"})
public class Reverse extends UDTF {
    @Override
    public void process(Object[] objects) throws UDFException {
        String str = (String) objects[0];
        String[] split = str.split(",");
        forward(split[0], split[1]);
    }
}

#END CODE;

SELECT foo('ab,dc') AS (a,b);

```

The return value of `@Resolve` must be of the `string[]` type. However, Janino-compiler cannot identify `"string->string,string"` in embedded code as `string[]`. To enable Janino-compiler to identify `"string->string,string"` as `string[]`, `"string->string,string"` must be enclosed in braces `{}`. When you create a Java UDTF by using a common method, braces `{}` are not required.

Define and call a Python code-embedded UDF

Sample code:

```

CREATE TEMPORARY FUNCTION foo AS 'embedded.UDFTest' USING
#CODE ('lang'='PYTHON', 'filename'='embedded')
from odps.udf import annotate
@annotate("bigint->bigint")
class UDFTest(object):
    def evaluate(self, a):
        return a * a
#END CODE;

SELECT foo(4);

```

- The indentation of Python code must comply with Python language specifications.
- When you register a Python UDF, the class name that follows the `AS` clause must contain the file name of the Python source code. You can use `'filename'='embedded'` to specify a virtual file name.

3.16.6. SQL functions

SQL functions allow you to reference SQL user-defined functions (UDFs) in SQL scripts. This topic describes how to manage SQL functions.

Background information

SQL functions of MaxCompute resolve the following issues:

- Generally, a large amount of similar code exists, which is inconvenient to maintain and prone to errors. To use UDFs, you must develop code, compile the code in Java, and then create resources and functions. The process is complicated. In addition, UDFs cannot catch up with built-in functions in terms of performance. Example:

```
SELECT
  NVL(STR_TO_MAP(GET_JSON_OBJECT(col, '$.key1')), 'default') AS key1,
  NVL(STR_TO_MAP(GET_JSON_OBJECT(col, '$.key2')), 'default') AS key2,
  ...
  NVL(STR_TO_MAP(GET_JSON_OBJECT(col, '$.keyN')), 'default') AS keyN
FROM t;
```

- One function can be transferred as a parameter to another only by using code similar to lambda expressions in Java.

Features

As a type of UDFs, SQL functions allow you to create UDFs in SQL and use function type parameters and anonymous functions. In this way, you can define business logic more flexibly. You can use SQL functions to simplify feature implementation and improve code reuse. SQL functions provide the following features:

- SQL functions allow you to reference and call SQL UDFs in SQL scripts.
- You can specify built-in functions, UDFs, or SQL functions in function type parameters when you call SQL functions.
- You can specify anonymous functions in function type parameters when you call SQL functions.

Create a permanent SQL function

After you create a permanent SQL function and store it in the MaxCompute meta system, all query statements can reference this function. To reference an SQL function in a query statement, you must be granted relevant permissions on the function. For more information, see [Authorize users](#) and [Grant access to a specific UDF to a specified user](#). Use the following syntax to create a permanent SQL function:

```
CREATE SQL FUNCTION function_name(@parameter_in1 datatype[, @parameter_in2 datatype...])
[RETURNS @parameter_out datatype]
AS [BEGIN]
function_expression
[END];
```

- **function_name**: the name of the SQL function to create. The name must be unique in the system.
- **parameter_in**: the input parameters of the SQL function.
- **RETURNS**: the variable to be returned by the SQL function. If you do not specify RETURNS, the value of function_name is returned by default.
- **parameter_out**: the output parameters of the SQL function.
- **function_expression**: the expression of the SQL function.

Example:

```
CREATE SQL FUNCTION MY_ADD(@a BIGINT) AS @a + 1;
```

In the preceding example, `@a + 1` indicates the implementation logic of the SQL function. You can write it as an expression to specify a built-in operator, built-in function, or UDF.

If the implementation logic is complex, you can write multiple SQL statements and enclose them with BEGIN and END. RETURNS specifies the variable to be returned by the SQL function. If you do not specify RETURNS, the value of function_name is returned by default. Example:

```
CREATE SQL FUNCTION MY_SUM(@a BIGINT, @b BIGINT, @c BIGINT) RETURNS @my_sum BIGINT
AS BEGIN
    @temp := @a + @b;
    @my_sum := @temp + @c;
END;
```

 **Note** To write multiple SQL statements, you must use Script Mode SQL. For more information, see [Script Mode SQL](#).

Query an SQL function

You can query an SQL function in the same way as querying a Java or Python UDF. Example:

```
DESC FUNCTION my_add;
```

 **Note** To query an SQL function on a client, make sure that the client version is later than 0.34.0.

Delete an SQL function

You can delete an SQL function in the same way as deleting a Java or Python UDF. Example:

```
DROP FUNCTION my_add;
```

Call an SQL function

You can call an SQL function in the same way as calling a built-in function. Example:

```
SELECT my_sum(col1, col2 ,col3) from t;
```

Define a temporary SQL function

If you only need to call an SQL function in a specific SQL script, you can define a temporary SQL function in the script. The temporary SQL function is not stored in the MaxCompute meta system, and applies only to the current SQL script. Example:

```
FUNCTION MY_ADD(@a BIGINT) AS @a + 1;
SELECT MY_ADD(key), MY_ADD(value) FROM src;
```

 **Note** To write multiple SQL statements, you must use Script Mode SQL. For more information, see [Script Mode SQL](#).

Use function type parameters

You can specify built-in functions, UDFs, or SQL functions in function type parameters when you call SQL functions. Example:

```
FUNCTION ADD(@a BIGINT) AS @a + 1;
FUNCTION OP(@a BIGINT, @fun FUNCTION (BIGINT) RETURNS BIGINT) AS @fun(@a);
SELECT OP(key, ADD), OP(key, ABS) FROM VALUES (1),(2) AS t (key);
```

-- Output

```
+-----+-----+
|_c0   |_c1   |
+-----+-----+
| 2     | 1     |
| 3     | 2     |
+-----+-----+
```

In the example, two input parameters are specified for the OP function. The `@a` parameter specifies a value of the BIGINT type. The `@fun` parameter specifies a function whose input and output parameters are both of the BIGINT type. The OP function transfers `@a` to the function that is specified by `@fun` and then to the ADD and ABS functions for processing. For more information about the ABS function, see [Mathematical functions](#).

Use anonymous functions

You can specify anonymous functions in function type parameters when you call SQL functions. Example:

```
FUNCTION OP(@a BIGINT, @fun FUNCTION (BIGINT) RETURNS BIGINT) AS @fun(@a);
SELECT OP(key, FUNCTION (@a) AS @a + 1) FROM VALUES (1),(2) AS t (key);
```

In the example, `FUNCTION (@a) AS @a + 1` is an anonymous function. You do not need to specify a data type for the input parameter `@a`. The compiler will infer the data type of `@a` based on the parameter definition of the OP function.

3.16.7. Open source geospatial UDFs

This topic describes how to use open source geospatial UDFs for spatial data analysis.

Apache Hive provides a set of open source geospatial UDFs. For more information, see [Github](#). MaxCompute provides native support for Hive UDFs. Therefore, you can use Hive geospatial functions in MaxCompute.

 **Note** If you encounter any issues when you use UDFs, submit the issues on GitHub for help.

Prepare local functions

1. In the **Command Prompt**, run the following command to download the geospatial UDFs for Hive 2.1.0 (corresponding to Hadoop 2.7.2) to your local device:

```
git clone -b "v2.1.0" --single-branch git@github.com:Esri/spatial-framework-for-hadoop.git
```

2. Use Maven to create a project.

```
cd spatial-framework-for-hadoop
mvn clean package -DskipTests -P java-8,hadoop-2.7,hive-2.1
```

3. Copy the created JAR package. This JAR package contains all methods of the open source geospatial UDFs.

```
cd ..
cp spatial-framework-for-hadoop/hive/target/spatial-sdk-hive-2.1.1-SNAPSHOT.jar ./spatial-sdk-hive.jar
```

4. Run the following command to download the JAR packages necessary for the project:

```
wget https://repo1.maven.org/maven2/com/esri/geometry/esri-geometry-api/2.2.0/esri-geometry-api-2.2.0.jar ./esri-geometry-api.jar
-- The preceding command is equivalent to the following command:
cp ~/.m2/repository/com/esri/geometry/esri-geometry-api/2.2.0/esri-geometry-api-2.2.0.jar ./esri-geometry-api.jar
```

Register functions in MaxCompute

1. In the MaxCompute **Client**, run the following commands to upload the two dependent JAR packages to the target project:

```
ADD JAR esri-geometry-api.jar;
ADD JAR spatial-sdk-hive.jar;
```

For more information about how to use the ADD command, see [Add a resource](#).

2. Run the following commands to register the functions:

```
CREATE FUNCTION ST_Aggr_ConvexHull AS 'com.esri.hadoop.hive.ST_Aggr_ConvexHull' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Aggr_Intersection AS 'com.esri.hadoop.hive.ST_Aggr_Intersection' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Aggr_Union AS 'com.esri.hadoop.hive.ST_Aggr_Union' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Area AS 'com.esri.hadoop.hive.ST_Area' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_AsBinary AS 'com.esri.hadoop.hive.ST_AsBinary' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_AsGeoJson AS 'com.esri.hadoop.hive.ST_AsGeoJson' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_AsJson AS 'com.esri.hadoop.hive.ST_AsJson' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_AsShape AS 'com.esri.hadoop.hive.ST_AsShape' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_AsText AS 'com.esri.hadoop.hive.ST_AsText' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Bin AS 'com.esri.hadoop.hive.ST_Bin' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_BinEnvelope AS 'com.esri.hadoop.hive.ST_BinEnvelope' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Boundary AS 'com.esri.hadoop.hive.ST_Boundary' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Buffer AS 'com.esri.hadoop.hive.ST_Buffer' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Centroid AS 'com.esri.hadoop.hive.ST_Centroid' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Contains AS 'com.esri.hadoop.hive.ST_Contains' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_ConvexHull AS 'com.esri.hadoop.hive.ST_ConvexHull' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
```

```

CREATE FUNCTION ST_CoordDim AS 'com.esri.hadoop.hive.ST_CoordDim' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Crosses AS 'com.esri.hadoop.hive.ST_Crosses' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Difference AS 'com.esri.hadoop.hive.ST_Difference' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Dimension AS 'com.esri.hadoop.hive.ST_Dimension' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Disjoint AS 'com.esri.hadoop.hive.ST_Disjoint' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Distance AS 'com.esri.hadoop.hive.ST_Distance' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_EndPoint AS 'com.esri.hadoop.hive.ST_EndPoint' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Envelope AS 'com.esri.hadoop.hive.ST_Envelope' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_EnvIntersects AS 'com.esri.hadoop.hive.ST_EnvIntersects' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Equals AS 'com.esri.hadoop.hive.ST_Equals' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_ExteriorRing AS 'com.esri.hadoop.hive.ST_ExteriorRing' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeodesicLengthWGS84 AS 'com.esri.hadoop.hive.ST_GeodesicLengthWGS84' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeomCollection AS 'com.esri.hadoop.hive.ST_GeomCollection' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Geometry AS 'com.esri.hadoop.hive.ST_Geometry' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeometryN AS 'com.esri.hadoop.hive.ST_GeometryN' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeometryType AS 'com.esri.hadoop.hive.ST_GeometryType' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeomFromGeoJson AS 'com.esri.hadoop.hive.ST_GeomFromGeoJson' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeomFromJson AS 'com.esri.hadoop.hive.ST_GeomFromJson' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeomFromShape AS 'com.esri.hadoop.hive.ST_GeomFromShape' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeomFromText AS 'com.esri.hadoop.hive.ST_GeomFromText' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_GeomFromWKB AS 'com.esri.hadoop.hive.ST_GeomFromWKB' USING 'spatial-

```

```

CREATE FUNCTION ST_GeomFromWKB AS 'com.esri.hadoop.hive.ST_GeomFromWKB' USING 'spatial-s
sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_InteriorRingN AS 'com.esri.hadoop.hive.ST_InteriorRingN' USING 'spatial-s
dk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Intersection AS 'com.esri.hadoop.hive.ST_Intersection' USING 'spatial-sdk-
hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Intersects AS 'com.esri.hadoop.hive.ST_Intersects' USING 'spatial-sdk-hive
.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Is3D AS 'com.esri.hadoop.hive.ST_Is3D' USING 'spatial-sdk-hive.jar,esri-ge
ometry-api.jar';
CREATE FUNCTION ST_IsClosed AS 'com.esri.hadoop.hive.ST_IsClosed' USING 'spatial-sdk-hive.jar
,esri-geometry-api.jar';
CREATE FUNCTION ST_IsEmpty AS 'com.esri.hadoop.hive.ST_IsEmpty' USING 'spatial-sdk-hive.jar,e
sri-geometry-api.jar';
CREATE FUNCTION ST_IsMeasured AS 'com.esri.hadoop.hive.ST_IsMeasured' USING 'spatial-sdk-h
ive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_IsRing AS 'com.esri.hadoop.hive.ST_IsRing' USING 'spatial-sdk-hive.jar,esri
-geometry-api.jar';
CREATE FUNCTION ST_IsSimple AS 'com.esri.hadoop.hive.ST_IsSimple' USING 'spatial-sdk-hive.jar,
esri-geometry-api.jar';
CREATE FUNCTION ST_Length AS 'com.esri.hadoop.hive.ST_Length' USING 'spatial-sdk-hive.jar,es
ri-geometry-api.jar';
CREATE FUNCTION ST_LineFromWKB AS 'com.esri.hadoop.hive.ST_LineFromWKB' USING 'spatial-sd
k-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_LineString AS 'com.esri.hadoop.hive.ST_LineString' USING 'spatial-sdk-hiv
e.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_M AS 'com.esri.hadoop.hive.ST_M' USING 'spatial-sdk-hive.jar,esri-geome
try-api.jar';
CREATE FUNCTION ST_MaxM AS 'com.esri.hadoop.hive.ST_MaxM' USING 'spatial-sdk-hive.jar,esri-
geometry-api.jar';
CREATE FUNCTION ST_MaxX AS 'com.esri.hadoop.hive.ST_MaxX' USING 'spatial-sdk-hive.jar,esri-g
eometry-api.jar';
CREATE FUNCTION ST_MaxY AS 'com.esri.hadoop.hive.ST_MaxY' USING 'spatial-sdk-hive.jar,esri-
geometry-api.jar';
CREATE FUNCTION ST_MaxZ AS 'com.esri.hadoop.hive.ST_MaxZ' USING 'spatial-sdk-hive.jar,esri-
geometry-api.jar';
CREATE FUNCTION ST_MinM AS 'com.esri.hadoop.hive.ST_MinM' USING 'spatial-sdk-hive.jar,esri-g
eometry-api.jar';
CREATE FUNCTION ST_MinX AS 'com.esri.hadoop.hive.ST_MinX' USING 'spatial-sdk-hive.jar,esri-g
eometry-api.jar';
CREATE FUNCTION ST_MinY AS 'com.esri.hadoop.hive.ST_MinY' USING 'spatial-sdk-hive.jar,esri-g

```



```

eometry-api.jar';
CREATE FUNCTION ST_MinZ AS 'com.esri.hadoop.hive.ST_MinZ' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MLineFromWKB AS 'com.esri.hadoop.hive.ST_MLineFromWKB' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MPointFromWKB AS 'com.esri.hadoop.hive.ST_MPointFromWKB' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MPolyFromWKB AS 'com.esri.hadoop.hive.ST_MPolyFromWKB' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MultiLineString AS 'com.esri.hadoop.hive.ST_MultiLineString' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MultiPoint AS 'com.esri.hadoop.hive.ST_MultiPoint' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_MultiPolygon AS 'com.esri.hadoop.hive.ST_MultiPolygon' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_NumGeometries AS 'com.esri.hadoop.hive.ST_NumGeometries' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_NumInteriorRing AS 'com.esri.hadoop.hive.ST_NumInteriorRing' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_NumPoints AS 'com.esri.hadoop.hive.ST_NumPoints' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Overlaps AS 'com.esri.hadoop.hive.ST_Overlaps' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Point AS 'com.esri.hadoop.hive.ST_Point' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_PointFromWKB AS 'com.esri.hadoop.hive.ST_PointFromWKB' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_PointN AS 'com.esri.hadoop.hive.ST_PointN' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_PointZ AS 'com.esri.hadoop.hive.ST_PointZ' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_PolyFromWKB AS 'com.esri.hadoop.hive.ST_PolyFromWKB' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Polygon AS 'com.esri.hadoop.hive.ST_Polygon' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Relate AS 'com.esri.hadoop.hive.ST_Relate' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_SetSRID AS 'com.esri.hadoop.hive.ST_SetSRID' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_SRID AS 'com.esri.hadoop.hive.ST_SRID' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';

```

```
CREATE FUNCTION ST_StartPoint AS 'com.esri.hadoop.hive.ST_StartPoint' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_SymmetricDiff AS 'com.esri.hadoop.hive.ST_SymmetricDiff' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Touches AS 'com.esri.hadoop.hive.ST_Touches' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Union AS 'com.esri.hadoop.hive.ST_Union' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Within AS 'com.esri.hadoop.hive.ST_Within' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_X AS 'com.esri.hadoop.hive.ST_X' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Y AS 'com.esri.hadoop.hive.ST_Y' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
CREATE FUNCTION ST_Z AS 'com.esri.hadoop.hive.ST_Z' USING 'spatial-sdk-hive.jar,esri-geometry-api.jar';
```

Test a UDF

Submit an SQL statement in the MaxCompute client (odpscmd) to test whether a UDF functions properly.

```
-- Enable Hive compatibility and submit a UDF for testing.
set odps.sql.hive.compatible=true;
SELECT ST_AsText(ST_Point(1, 2));
```

The following response indicates that the UDF package is installed:

```
+-----+
| _c0 |
+-----+
| POINT (1 2) |
+-----+
```

3.16.8. UDTF usage

This topic describes how to use Java user-defined table-valued functions (UDTFs) and Python UDTFs.

Usage

In typical cases, a UDTF is used as follows in Structured Query Language (SQL):

```
select user_udtf(col0, col1, col2) as (c0, c1) from my_table;  
select user_udtf(col0, col1, col2) as (c0, c1) from (select * from my_table distribute by key sort by key) t  
;  
select reduce_udtf(col0, col1, col2) as (c0, c1) from (select col0, col1, col2 from (select map_udtf(a0, a1,  
a2, a3) as (col0, col1, col2) from my_table) t1 distribute by col0 sort by col0, col1) t2;
```

UDTFs are subject to the following limits:

- A select clause cannot contain any other expressions.

```
select value, user_udtf(key) as mycol ...
```

- A UDTF cannot be nested.

```
select user_udtf1(user_udtf2(key)) as mycol...
```

- A select clause cannot be used with a group by, distribute by, or sort by clause.

```
select user_udtf(key) as mycol ... group by mycol
```

3.17. UDT

3.17.1. UDT

MaxCompute has introduced the User-defined type (UDT) based on the new generation SQL engine. UDT allows you to reference classes or objects of third-party languages in SQL statements to obtain data or call methods.

Scenarios

UDT are typically applied in the following scenarios:

- Scenario 1: MaxCompute does not have built-in functions to complete tasks that can be easily performed using other languages. For example, some tasks can be performed by calling built-in Java classes only once. The procedure of using user defined functions (UDFs) to complete these tasks is complex.
- Scenario 2: You need to call a third-party library in SQL statements to implement the corresponding function. You want to use a function provided by a third-party library directly in a SQL statement, instead of wrapping the function inside a UDF.
- Scenario 3: Select Transform allows you to include objects and classes in SQL statements to make these SQL statements easier to read and maintain. For some languages, such as Java, the source code can be executed only after it is compiled. You want to reference objects and classes of these languages in SQL statements.

Overview

UDT allows you to reference classes or objects of third-party languages in SQL statements to obtain data or call methods.

There are major differences between UDT supported by MaxCompute and other SQL engines. UDT supported by other SQL engines are similar to the struct composite type in MaxCompute. Some proprietary SQL languages provide features that allow you to call third-party libraries, such as the CREATE TYPE statement in Oracle databases. UDT supported by MaxCompute is similar to the CREATE TYPE statement. A UDT contains both fields and methods. In addition, MaxCompute does not require that you use Data Definition Language (DDL) statements to define type mappings. Instead, MaxCompute allows you to reference types directly in SQL statements.

Example:

```
set odps.sql.type.system.odps2=true;
-- Enable new data type support in MaxCompute. The following example will use a new type of Integer
(int).
SELECT java.lang.Integer.MAX_VALUE;
```

The output is as follows:

```
+-----+
| max_value |
+-----+
| 2147483647 |
+-----+
```

You can shorten the statement by removing java.lang in the same way as in Java:

```
set odps.sql.type.system.odps2=true;
SELECT Integer.MAX_VALUE;
```

The expression in the preceding SELECT statement is similar to a Java expression and is executed in the same way as in Java. The expression specifies a UDT in MaxCompute.

You can use UDF to implement all functions provided by UDT. If you use a UDF to implement the same function, you need to follow these steps:

1. Define a UDF class.

```
package com.aliyun.odps.test;

public class IntegerMaxValue extends com.aliyun.odps.udf.UDF {

    public Integer evaluate() {
        return Integer.MAX_VALUE;
    }

}
```

2. Compile the UDF and pack it into a JAR package. Upload the JAR package and create a function.

```
add jar odps-test.jar;
create function integer_max_value as 'com.aliyun.odps.test.IntegerMaxValue' using 'odps-test.jar'
;
```

3. Call the function in a SQL statement.

```
select integer_max_value();
```

Using a UDT simplifies this procedure. By using UDT, you can use external functions provided by other languages in SQL statements more easily.

How UDT works

The example shows how to use UDT to access Java static fields. You can use UDT to implement more functions. The following example shows the UDT execution procedure and its functions.

```
-- Sample data
@table1 := select * from values ('10000000000000000000') as t(x);
@table2 := select * from values (100L) as t(y);
-- Logic of the code
@a := select new java.math.BigInteger(x) x from @table1;      -- Create a new object
@b := select java.math.BigInteger.valueOf(y) y from @table2;  -- Call a static method.
select /*+mapjoin(b)*/ x.add(y).toString() from @a a join @b b; -- Call an instance method
```

The output is follows:

```
10000000000000000000100
```

This example also shows how to retrieve UDT columns using subqueries, which is difficult to complete using UDF. The x column retrieved by variable a is java.math.BigInteger type. It is not a built-in type. You can pass the UDT data to another operator and then call its method. You can also use the UDT data in data shuffling.

□

This figure shows that a UDT has three stages: M1, R2, and J3. When a Join clause is used, data reshuffling is required, which is the same as MapReduce. Data is processed at multiple stages. Typically, data processing at different stages are performed in different processes or different physical machines.

Only the new java.math.BigInteger(x) method is called at the M1 stage.

The java.math.BigInteger.valueOf(y) and x.add(y).toString() methods are called separately at the J3 stage. These methods are called at different stages in different processes or physical machines. The UDT encapsulates these stages and acts as a JVM.

Features

- UDT currently only supports Java. Other languages will be supported in later versions.
- UDT also allows you to upload JAR packages and directly reference the packages. UDT has provided flags.

- You can use `set odps.sql.session.resources` to specify one or more resources that you need to reference and separate the resources with commas (,). Example: `set odps.sql.session.resources=foo.sh,bar.txt; .`

 **Note** This flag is the same as the resource setting flag in Select Transform. Therefore, this flag controls two functions. For example, you can use a UDT to reference the UDF JAR package that we have mentioned in the Overview section.

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.resources=odps-test.jar;
--To reference the JAR package, you must first upload the package to the corresponding project and make sure that it is a JAR type resource.
select new com.aliyun.odps.test.IntegerMaxValue().evaluate();
```

- You can use `odps.sql.session.java.imports` to specify one or more default JAR packages separated with commas (,). It is similar to the Java import statement. You can specify a class path, such as `java.math.BigInteger`, or use `*`. Currently, `static import` is not supported.

For example, you can use a UDT to reference the UDF JAR package that we have mentioned in Overview.

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.resources=odps-test.jar;
set odps.sql.session.java.imports=com.aliyun.odps.test.*;
-- Specify the default JAR package.
select new IntegerMaxValue().evaluate();
```

- UDT supports the following operations:
 - Instantiate objects using the new operator.
 - Instantiate arrays using the new operator, including ArrayList initialization. Example: `new Integer[] { 1, 2, 3 }.`
 - Call methods, including static methods. You can create objects in the factory pattern.
 - Access fields, including static fields.

Note

- Only public methods and public fields are supported.
- Identifiers in UDT include package names, class names, method names, and field names. All identifiers are case-sensitive.
- UDT supports type conversion and SQL syntax, for example, `cast (1 as java.lang.Object)`. UDT does not support type conversion using Java syntax, for example, `(Object)1`.
- Anonymous classes and lambda expressions are not supported. They may be supported in later versions.
- UDT is typically used in expressions. Functions that do not return values cannot be called in expressions. Therefore, UDT currently does not support calling functions that do not return values. This issue will be resolved in a later version.

- By default, you can reference all classes provided by Java SDK.

Note The runtime currently uses JDK1.8. Later versions may be not supported.

- All operators use the MaxCompute SQL semantics. The result of `String.valueOf(1) + String.valueOf(2)` is 3. The two strings are implicitly converted to double type values and summed. If you use Java string concatenation to merge the strings, the result will be 12.

In addition to the string concatenation methods in MaxCompute and Java, you may also have confusion about the equal (=) operator. The equal (=) operator in SQL statements is used as a comparison operator. To compare two Java objects, you must call the equals method. You cannot use the equal (=) operator to verify whether two objects are equal. When UDT are used, you cannot guarantee that one object is equal to another object. For more information, see the following descriptions.

- Java data types are mapped to built-in data types. For more information, see the data type mapping table in [Java UDFs](#). The mapping table can be applied to UDT.
 - Built-in type data can directly call the method of the Java type to which the built-in type is mapped. Example: `'123'.length()` , `1L.hashCode()` .
 - UDT can be used in built-in functions and UDF. For example, in `chr(Long.valueOf('100'))` , `Long.valueOf` returns a `java.lang.Long` type value. Built-in function `chr` supports the built-in type of `BIGINT`.
 - Java primitive type data is automatically converted to boxing type data and the preceding two rules can be applied in this situation.

Note For certain built-in **new data types**, you must add the `set odps.sql.type.system.odps2=true;` statement to declare these types. Otherwise, an error occurs.

- UDT completely support Java generics. For example, based on the parameter type, the compiler knows the type of the value returned by the `java.util.Arrays.asList(new java.math.BigInteger('1'))` method is `java.util.List<java.math.BigInteger>` .

Note You must set the type parameter in a construct function or use `java.lang.Object`, which is the same as in Java. For example, the result of `new java.util.ArrayList(java.util.Arrays.asList('1', '2'))` is `java.util.ArrayList<Object>`. The result of `new java.util.ArrayList<String>(java.util.Arrays.asList('1', '2'))` is `java.util.ArrayList<String>`.

- UDT does not have a clear definition of equal objects. This is caused by data reshuffling. The join example shows that objects may be transmitted between different processes or physical machines. During the transmission, an object may be referenced as two different objects. For example, an object may be shuffled to two machines and then reshuffled.

Therefore, when you use UDT, you must use the `equals` method to compare two objects instead of using the `equal (=)` operator.

The correlations between objects in a row or column can be guaranteed. However, the correlations between objects in different rows or columns are not guaranteed.

- Currently, UDT cannot be used as shuffle keys in the join, group by, distribute by, sort by, order by, and cluster by clauses.

UDT can be used at any stages in expressions, but cannot be output as final results. For example, you cannot call the `group by new java.math.BigInteger('123')` method. Instead, you can call the `group by new java.math.BigInteger('123').hashCode()` method. This is because the value returned by `hashCode` is an `int.class` type, which can be used as a built-in type of `int`. This applies the built-in type to Java type mapping rules.

- UDT have made the data type conversion rules more flexible:
 - UDT objects can be converted to objects of its base classes by implicit conversion.
 - UDT objects can be forcibly converted to objects of its base classes or subclasses.
 - Data type conversion for two objects without inheritance applies the native conversion rules. The conversion may change the data. For example, `java.lang.Long` type data can be forcibly converted to `java.lang.Integer` type data. It is a process of converting built-in `BIGINT` type data to `INT` type data. This process may cause data changes or even data precision changes.
- Currently, you cannot save UDT objects, which means that you cannot add UDT objects to tables. DDL does not support UDT. You cannot create tables that contain UDT, unless you convert the data to built-in types using implicit conversion. In addition, the final output cannot be UDT type. To resolve this issue, you can call the `toString()` method to convert the UDT data to `java.lang.String` type data because all Java classes support the `toString()` method. You can use this method to check UDT data during debugging.

You can also add the `set odps.sql.udt.display.tostring=true;` statement to enable MaxCompute to convert all UDT data to strings by calling the `java.util.Objects.toString(...)` method for debugging. This flag is typically used for debugging because it can only be applied to the print statement. It cannot be applied to the insert statement.

Binary is a built-in type and supports auto serialization. You can then save the `byte[]` arrays. The saved `byte[]` arrays can be deserialized to binary type.

Some classes may have their own serialization and deserialization methods, such as `protobuf`. To save UDT, you must call serialization and deserialization methods to convert the data to binary data.

- You can use UDT to achieve the function provided by the scalar function. With the built-in functions `collect list` and `explode`, you can use UDT to achieve the functions provided by the aggregator and table functions.

UDT examples

- Example of using Java arrays

```
set odps.sql.type.system.odps2=true;
set odps.sql.udt.display.toString=true;
select
  new Integer[10], -- Create an array that contains 10 elements.
  new Integer[] {c1, c2, c3}, -- Create an array that contains three elements by initializing an ArrayList.
  new Integer[][] { new Integer[] {c1, c2}, new Integer[] {c3, c4} }, -- Create a multidimensional array.
  new Integer[] {c1, c2, c3} [2], -- Access the elements in the array using indexes.
  java.util.Arrays.asList(c1, c2, c3); -- This is another way to create a built-in array. It creates a List<Integer>, which can be used as an array<int>.
from values (1,2,3,4) as t(c1, c2, c3, c4);
```

- Example of using JSON

The runtime of UDT carries a JSON dependency (version 2.2.4), which can be directly used in JSON.

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*,java.com.google.gson.*; -- To import multiple packages, separate the packages with commas (,).
@a := select new Gson() gson; -- Create a gson object.
select
  gson.toJson(new ArrayList<Integer>(Arrays.asList(1, 2, 3))), -- Convert an object to a JSON string.
  cast(gson.fromJson('["a","b","c"]', List.class) as List<String>) --Deserialize the JSON string. The gson also forcibly converts the deserialized result from List<Object> type to List<String> type.
from @a;
```

Compared with built-in function `get_json_object`, this method is simple and it improves efficiency by extracting content from the JSON string and then deserializing the string to a supported data type.

In addition to JSON dependencies, MaxCompute runtime also carries other dependencies, including commons-logging (1.1.1), commons-lang (2.5), commons-io (2.4), and protobuf-java (2.4.1).

- Example of using composite types

Built-in types of array and map are mapped to `java.util.List` and `java.util.Map`, respectively. Results:

- Java objects in classes calling the `java.util.List` or `java.util.Map` interface can be used in MaxCompute SQL composite type data processing.

- Array and map type data in MaxCompute can directly call the List or Map interface.

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*;
select
    size(new ArrayList<Integer>()),    -- Call built-in function size to obtain the size of the ArrayList
    .
    array(1,2,3).size(),              -- Call the List method for built-in type array.
    sort_array(new ArrayList<Integer>()), -- Sort the data in the ArrayList.
    al[1],                            -- The Java List method does not support indexing. However, the array type supports indexing.
    Objects.toString(a),              -- With this method, you can now convert array type to string type data.
    array(1,2,3).subList(1, 2)        -- Get a sublist.
from (select new ArrayList<Integer>(array(1,2,3)) as al, array(1,2,3) as a) t;
```

- Example of aggregation

To use UDT to achieve aggregation, you must first use built-in function `collect_set` or `collect_list` to convert the data to the List type, and then call the UDT methods to aggregate the data.

The following example shows how to obtain the median from BigInteger data. You cannot directly call the built-in `median` function because the data is `java.math.BigInteger` type.

```
set odps.sql.session.java.imports=java.math.*;
@test_data := select * from values (1),(2),(3),(5) as t(value);
@a := select collect_list(new BigInteger(value)) values from @test_data; -- Aggregate the data to a list.
@b := select sort_array(values) as values, values.size() cnt from @a; -- To obtain the median, first sort the data.
@c := select if(cnt % 2 == 1, new BigDecimal(values[cnt div 2]), new BigDecimal(values[cnt div 2 - 1].add(values[cnt div 2])).divide(new BigDecimal(2))) med from @b;
-- Final output
select med.toString() from @c;
```

You cannot use the `collect_list` function to implement partial aggregation because it aggregates all data. Using the built-in aggregator or UDAF object is more efficient. We recommend that you use the built-in aggregator if possible. Aggregating all data in a group increases the risk of data skew.

If the logic of the UDAF object is to aggregate all data, which is the same as the built-in function `wm_concat`, using the `collect_list` function is more efficient than using the UDAF object.

- Example of using the table-valued function

The table-valued function allows you to input and output multiple rows and columns. To input or output multiple rows and columns, follow these steps:

- To input multiple rows or columns, reference the example of using the aggregate

function.

- ii. To output multiple rows, you can use a UDT to define a Collection type (List or Map), and then call the explode function to split the collection into multiple rows.
- iii. A UDT can contain multiple fields. You can retrieve the data from the fields by calling different getter methods. The data is then output in multiple rows.

The following example shows how to split a JSON string and output the splitting result in multiple columns:

```
@a := select '["a":"1","b":"2"],["a":"1","b":"2"]' str; -- Sample data
@b := select new com.google.gson.Gson().fromJson(str, java.util.List.class) l from @a; -- Deserialize
the JSON string.
@c := select cast(e as java.util.Map<Object,Object>) m from @b lateral view explode(l) t as e; -- Call
the explode function to split the string.
@d := select m.get('a') as a, m.get('b') as b from @c; -- Output the splitting result in multiple columns
.
select a.toString() a, b.toString() b from @d; -- The final output. Columns a and b in variable d are O
bject type.
```

Features, performance, and security

UDT has the following features:

- Easy to use. You do not need to define any functions.
- You can directly use all functions supported by the JDK. This improves the flexibility of SQL.
- You can directly reference objects and classes of other languages in SQL statements, which is easy to manage.
- You can directly reference libraries of other languages. This enables you to reuse your code.
- You can achieve functions based on the object-oriented concept.

Improvements that will be made to UDT later:

- UDT will support functions that do not return values, including functions that return values but the returned values are ignored and the passed in data is used, such as the add method provided by the List interface. This method returns the list that you have passed in.
- Anonymous classes and lambda expressions will be supported.
- You can use UDT as shuffle keys.
- More languages will be supported, such as Python.

The execution procedure of UDT is similar to UDF. UDT and UDFs have almost the same performance. In addition, the compute engine has been greatly improved. Therefore, UDT has higher performance in certain scenarios.

- Deserialization is not required for objects in only one process. Deserialization is required only when the objects are transmitted among processes. This means that UDT do not incur any serialization or deserialization overhead when no data reshuffling is performed, such as calling the join or aggregator function.
- The runtime of UDT is based on Codegen. It is not based on reflection. Therefore, no performance reduction is experienced. Multiple UDT is wrapped in one FunctionCall and executed at the same time. For example, you may think that multiple UDT methods are called

in `values[x].add(values[y]).divide(java.math.BigInteger.valueOf(2))` , but actually only one method is called. UDT focus on small-granularity data processing. However, this does not incur additional overhead for the interface where multiple functions are called.

UDT is restricted by the [Java sandbox](#) model, as same as UDF. To perform restricted operations, you must enable sandbox isolation or apply to join a sandbox whitelist.

3.17.2. Usage examples

This topic describes common usage examples of user-defined type (UDT). For example, you can use Java arrays, JSON, complex data types, and table-valued functions in UDT. You can also aggregate data, overload functions, and reference embedded code in UDT.

Use Java arrays

```
set odps.sql.type.system.odps2=true;
set odps.sql.udt.display.toString=true;
SELECT
    new Integer[10], -- Create an array that contains 10 elements.
    new Integer[] {c1, c2, c3}, -- Create an array that contains three elements by initializing an ArrayList.
    new Integer[][] { new Integer[] {c1, c2}, new Integer[] {c3, c4} }, -- Create a multidimensional array.
    new Integer[] {c1, c2, c3} [2], -- Access the elements in the array by using indexes.
    java.util.Arrays.asList(c1, c2, c3); -- Create a list of the List<Integer> type, which can be used as a
    n array of the Array<Int> type. This is another way to create a built-in array.
FROM VALUES (1,2,3,4) AS t(c1, c2, c3, c4);
```

Use JSON

The runtime of UDT carries a JSON dependency (version 2.2.4), which can be directly used in JSON.

 **Note** In addition to JSON dependencies, MaxCompute runtime also carries other dependencies, including `commons-logging (1.1.1)` , `commons-lang (2.5)` , `commons-io (2.4)` , and `protobuf-java (2.4.1)` .

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*,java.com.google.gson.*; -- To import multiple packages at
a time, separate the packages with commas (,).
@a := select new Gson() gson; -- Create a Gson object.
select
    gson.toJson(new ArrayList<Integer>(Arrays.asList(1, 2, 3))), -- Convert an object to a JSON string.
    cast(gson.fromJson('["a","b","c"]', List.class) as List<String>) -- Deserialize the JSON string. Gson forcibl
y converts the deserialization result from the List<Object> type to the List<String> type.
from @a;
```

Compared with the built-in function `GET_JSON_OBJECT`, this UDT-based method is simpler. In addition, this method deserializes the content that is extracted from the JSON string to a supported data type, thereby improving efficiency.

Use complex data types

The built-in data types ARRAY and MAP map the `java.util.List` and `java.util.Map` methods, respectively.

- Java objects in classes that implement `java.util.List` or `java.util.Map` can be used in complex-type data processing in MaxCompute SQL.
- MaxCompute can directly call `java.util.List` or `java.util.Map` to process data of the ARRAY or MAP type.

```
set odps.sql.type.system.odps2=true;
set odps.sql.session.java.imports=java.util.*;
select
    size(new ArrayList<Integer>()),    -- Call the built-in function size() to obtain the size of the Array
    List.
    array(1,2,3).size(),              -- Call java.util.List to process data of the ARRAY type.
    sort_array(new ArrayList<Integer>()), -- Sort the data in the ArrayList.
    al[1],                            -- java.util.List does not support indexing but can process data of the ARRAY
    type that supports indexing.
    Objects.toString(a),              -- Convert data from the ARRAY type to the STRING type.
    array(1,2,3).subList(1, 2)        -- Obtain a sublist.
from (select new ArrayList<Integer>(array(1,2,3)) as al, array(1,2,3) as a) t;
```

Aggregate data

To use UDT to achieve aggregation, you must first use the built-in function `COLLECT_SET` or `COLLECT_LIST` to aggregate the data to a list, and then call the scalar method of UDT to calculate the aggregate value.

The following example shows how to calculate the median of BigInteger data. You cannot directly call the built-in function `MEDIAN` because the data is of the `java.math.BigInteger` type.

```
set odps.sql.session.java.imports=java.math.*;
@test_data := select * from values (1),(2),(3),(5) as t(value);
@a := select collect_list(new BigInteger(value)) values from @test_data; -- Aggregate the data to a list.
@b := select sort_array(values) as values, values.size() cnt from @a; -- Sort the data.
@c := select if(cnt % 2 == 1, new BigDecimal(values[cnt div 2]), new BigDecimal(values[cnt div 2 - 1]).add
(values[cnt div 2])).divide(new BigDecimal(2)) med from @b;
-- The final output.
select med.toString() from @c;
```

You cannot use the `collect_list` function to aggregate partial data because it can only aggregate all data in a specific group. It is less efficient than built-in aggregate functions of MaxCompute or user-defined aggregate functions (UDAFs). We recommend that you use built-in aggregate functions if possible. Aggregating all data in a group increases the risk of data skew.

The UDT-based method produces a higher efficiency than a built-in aggregate function such as `WM_CONCAT` in aggregating all data in a group.

Implement table-valued functions

Table-valued functions allow you to specify multiple input rows and columns, and can generate multiple output rows and columns. To implement a table-valued function, perform the following steps:

- Specify multiple input rows or columns. For more information, see [Aggregate data](#).
- Call the `java.util.List` or `java.util.Map` method to generate a data collection, and then call the `explode` function to split the collection into multiple rows.
- Call different getter methods to obtain data from different fields in UDT. The obtained data is returned in multiple columns.

The following example shows how to split a JSON string and return the splitting result in multiple columns:

```
@a := select '[{"a":"1","b":"2"}, {"a":"1","b":"2"}]' str; -- The sample data.
@b := select new com.google.gson.Gson().fromJson(str, java.util.List.class) l from @a; -- Deserialize the JSON string.
@c := select cast(e as java.util.Map<Object,Object>) m from @b lateral view explode(l) t as e; -- Call the explode function to split the string.
@d := select m.get('a') as a, m.get('b') as b from @c; -- Return the splitting result in multiple columns.
select a.toString() a, b.toString() b from @d; -- The final output. Columns a and b in the variable d are of the Object type.
```

Overload functions

In MaxCompute, user-defined functions (UDFs) overload functions by overloading the `evaluate` method. Generics are not supported in this mode. Therefore, you must write an `evaluate` method for each supported data type when you define a function. Functions with some input types such as `ARRAY` cannot be overloaded in this mode. If the `Resolve` annotation is unavailable, Python UDFs or user-defined table-valued functions (UDTFs) determine input parameters based on the number of parameters and support variable-length parameters. However, when this mechanism is used, the compiler does not detect specific errors in static mode.

In this case, you can use UDT to overload functions. UDT supports generics, class inheritance, and variable-length parameters, and offer flexible methods to define functions. Example:

```

public class UDTClass {
    // This function supports a numeric-type parameter and returns a value of the DOUBLE type. The pa
    rameter can be of the TINYINT, SMALLINT, INT, BIGINT, FLOAT, or DOUBLE type. You can also specify a u
    ser-defined data type that is defined based on a numeric type in the parameter.
    public static Double doubleValue(Number input) {
        return input.doubleValue();
    }
    // This function supports a numeric-type parameter and a parameter of any data type. The return v
    alue is of the same type as the second parameter.
    public static <T extends Number, R> R nullOrValue(T a, R b) {
        return a.doubleValue() > 0 ? b : null;
    }
    // This function supports a parameter of the ARRAY or LIST type. The parameter value can contain a
    ny type of elements. The function returns a value of the BIGINT type.
    public static Long length(java.util.List<? extends Object> input) {
        return input.size();
    }
    // If the code does not forcibly convert the data type, you must specify the default map object java.
    util.Map<Object, Object> of UDT. To transfer another map object, for example, map<bigint,bigint>, con
    sider using the following method:
    // 1. When you define the function, use java.util.Map<? extends Object, ? extends Object>.
    // 2. When you call the function, implement a cast function to forcibly convert the type, for example,
    UDTClass.mapSize(cast(mapObj as java.util.Map<Object, Object>)).
    public static Long mapSize(java.util.Map<Object, Object> input) {
        return input.size();
    }
}

```

In specific scenarios, a UDF can use `com.aliyun.odps.udf.ExecutionContext` that is transferred by the `setup` method to obtain the context. UDT can call the `com.aliyun.odps.udt.UDTExecutionContext.get()` method to obtain the `ExecutionContext` object.

Reference embedded code in UDT

Code-embedded UDFs allow you to place SQL scripts and third-party code lines in the same source code file. This simplifies the usage of UDT and facilitates daily development and maintenance. For more information, see [Code-embedded UDFs](#).

3.18. UDJ

MaxCompute introduces user defined join (UDJ) to the user defined function (UDF) framework based on the MaxCompute V2.0 compute engine. UDJ allows more customized operations by flexibly joining tables and simplifies MapReduce-based operations in the underlying distributed system.

UDJ

Overview

MaxCompute provides multiple **JOIN** methods, including **INNER JOIN**, **RIGHT JOIN**, **OUTER JOIN**, **LEFT JOIN**, **FULL JOIN**, **SEMI JOIN**, and **ANTI-SEMI JOIN**. You can use these built-in **JOIN** methods in most scenarios. However, these methods are insufficient if you want to perform operations on multiple tables.

In most cases, you can build your code framework by using UDFs. However, the current UDF, user-defined table-valued function (UDTF), and user-defined aggregation function (UDAF) frameworks can only handle one table at a time. If you want to perform user-defined operations on multiple tables, you must use built-in **JOIN** methods, **UDFs**, **UDTFs**, and complex **SQL** statements. In such scenarios, you have to use a custom MapReduce framework instead of **SQL** to complete the required computing tasks.

Regardless of the scenario, these operations require technological expertise and may cause the following issues:

- In scenarios where you use **built-in JOIN methods, UDFs, UDTFs, and complex SQL statements**: The use of multiple **JOIN** methods and code in **SQL** statements results in a logical black box, which causes difficulties in generating an optimal execution plan.
- In scenarios where you use a custom MapReduce framework: Execution plans are hard to optimize. Most of the MapReduce code is written in Java. During the deep optimization of native runtime code, the execution of the MapReduce code is less efficient than the execution of the MaxCompute code generated by the LLVM code generator.

UDJ performance

The following example uses a real online MapReduce job to test UDJ performance. This job uses a complex algorithm to join two tables. This example uses UDJ to rewrite the **SQL** statements of the job and then verify the execution result. Under the same data concurrency, the comparison between MapReduce and UDJ performance is as follows:

□

As shown in the preceding figure, UDJ helps describe the complex logic of handling multiple tables and greatly improves the execution performance. The code is only executed inside UDJ. The entire logic of the code, such as the logic of mapper in this example, is executed based on the highly efficient native runtime of MaxCompute. UDJ optimizes the data exchange logic between the MaxCompute runtime engine and Java interfaces. The **JOIN** logic of UDJ is more efficient than that of reducer.

Examples

The following example describes how to use UDJ in MaxCompute.

Assume that two tables named `payment` and `user_client_log` exist.

- The `payment` table stores the payment records of users. Each payment record contains the user ID, payment time, and payment content. The following table lists the sample data.

user_id	time	pay_info
2656199	2018-02-13 22:30:00	gZhvdYSOQb
8881237	2018-02-13 08:30:00	pYvotuLDIT
8881237	2018-02-13 10:32:00	KBuMzRpsko

- The user_client_log table stores the client logs of users. Each log contains the user ID, log time, and log content. The following table lists the sample data.

user_id	time	content
8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn
8881237	2018-02-13 06:14:00	click OkTYNUHMqZzIDyL
8881237	2018-02-13 10:30:00	click OkTYNUHMqZzIDyL

Requirement: For a log record in the user_client_log table, find the payment record that has the closest time to this log record in the payment table. Then, join the two records and generate the result. The following table lists the output result.

user_id	time	content
8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn, pay pYvotuLDIT
8881237	2018-02-13 06:14:00	click OkTYNUHMqZzIDyL, pay pYvotuLDIT
8881237	2018-02-13 10:30:00	click OkTYNUHMqZzIDyL, pay KBuMzRpsko

You can use one of the following methods to meet the preceding requirement:

- Use built-in JOIN methods. The SQL pseudocode is as follows:

```
SELECT
  p.user_id,
  p.time,
  merge(p.pay_info, u.content)
FROM
  payment p RIGHT OUTER JOIN user_client_log u
ON p.user_id = u.user_id and abs(p.time - u.time) = min(abs(p.time - u.time))
```

When you join two records in the tables, you must calculate the minimum difference between p.time and u.time under the same user_id. However, you cannot call aggregate functions in a JOIN condition. Therefore, you cannot use standard JOIN methods to complete this task.

- Use UDJ.
 - Register the UDJ function.

a. Configure the SDK of the new version.

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-udf</artifactId>
  <version>0.29.10-public</version>
  <scope>provided</scope>
</dependency>
```

b. Write the UDJ code and package the code as odps-udj-example.jar.

```
package com.aliyun.odps.udf.example.udj;
import com.aliyun.odps.Column;
import com.aliyun.odps.OdpsType;
import com.aliyun.odps.Yieldable;
import com.aliyun.odps.data.ArrayRecord;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.udf.DataAttributes;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.UDJ;
import com.aliyun.odps.udf.annotation.Resolve;
import java.util.ArrayList;
import java.util.Iterator;
/** For each record in the table on the right, find the record that has the closest time to this record.
 * Merge two records.
 */
@Resolve("->string,bigint,string")
public class PayUserLogMergeJoin extends UDJ {
  private Record outputRecord;
  /** Call the preceding code before data is processed. You can use the PayUserLogMergeJoin in method for initialization.
  */
  @Override
  public void setup(ExecutionContext executionContext, DataAttributes dataAttributes) {
    //
    outputRecord = new ArrayRecord(new Column[]{
      new Column("user_id", OdpsType.STRING),
      new Column("time", OdpsType.BIGINT),
      new Column("content", OdpsType.STRING)
    });
  }
}
```

```

/** Rewrite the PayUserLogMergeJoin method to implement the connection logic.
 * @param key: the current key for joining.
 * @param left: the record group of the current key in the table on the left.
 * @param right: the record group of the current key in the table on the right.
 * @param output: generates the output of UDJ.
 */
@Override
public void join(Record key, Iterator<Record> left, Iterator<Record> right, Yieldable<Record> output) {
    outputRecord.setString(0, key.getString(0));
    if (! right.hasNext()) {
        // The group on the right is empty. Do not perform operations on this group.
        return;
    } else if (! left.hasNext()) {
        // The group on the left is empty. Generate all records from the group on the right but do not merge the records of the two tables.
        while (right.hasNext()) {
            Record logRecord = right.next();
            outputRecord.setBigint(1, logRecord.getDatetime(0).getTime());
            outputRecord.setString(2, logRecord.getString(1));
            output.yield(outputRecord);
        }
        return;
    }
    ArrayList<Record> pays = new ArrayList<>();
    // The group on the left iterates from start to end.
    // The iterator cannot reset the records of the group on the right.
    // Therefore, save each record of the group on the left to ArrayList.
    left.forEachRemaining(pay -> pays.add(pay.clone()));
    while (right.hasNext()) {
        Record log = right.next();
        long logTime = log.getDatetime(0).getTime();
        long minDelta = Long.MAX_VALUE;
        Record nearestPay = null;
        // Iterate all the records of the group on the left. Then, you can find the minimum time delta between the records.
        for (Record pay: pays) {
            long delta = Math.abs(logTime - pay.getDatetime(0).getTime());
            if (delta < minDelta) {
                minDelta = delta;
                nearestPay = pay;
            }
        }
    }
}

```

```

    }
    }
    // Merge the log records and payment records that are closest in time and then generate results.
    outputRecord.setBigint(1, log.getDatetime(0).getTime());
    outputRecord.setString(2, mergeLog(nearestPay.getString(1), log.getString(1)));
    output.yield(outputRecord);
    }
}

String mergeLog(String payInfo, String logContent) {
    return logContent + ", pay " + payInfo;
}

@Override
public void close() {
}
}

```

- c. Add JAR package resources to MaxCompute.

```
add jar odps-udj-example.jar;
```

- d. Register the UDJ function *pay_user_log_merge_join* in MaxCompute.

```
create function pay_user_log_merge_join
as 'com.aliyun.odps.udf.example.udj.PayUserLogMergeJoin'
using 'odps-udj-example.jar';
```

- ii. Prepare sample data.

- a. This example creates the payment table and the user_client_log table.

```
create table payment(user_id string,time datetime,pay_info string);
create table user_client_log(user_id string,time datetime,content string);
```

- b. Insert data into the sample tables.

```

-- Insert data into the payment table.
INSERT OVERWRITE TABLE payment VALUES
('1335656', datetime '2018-02-13 19:54:00', 'PEqMSHyktn'),
('2656199', datetime '2018-02-13 12:21:00', 'pYvotuLDIT'),
('2656199', datetime '2018-02-13 20:50:00', 'PEqMSHyktn'),
('2656199', datetime '2018-02-13 22:30:00', 'gZhvdYSOQb'),
('8881237', datetime '2018-02-13 08:30:00', 'pYvotuLDIT'),
('8881237', datetime '2018-02-13 10:32:00', 'KBuMzRpsko'),
('9890100', datetime '2018-02-13 16:01:00', 'gZhvdYSOQb'),
('9890100', datetime '2018-02-13 16:26:00', 'MxONdLckwa')
;

-- Insert data into the user_client_log table.
INSERT OVERWRITE TABLE user_client_log VALUES
('1000235', datetime '2018-02-13 00:25:36', 'click FNOXAibRjklAQPB'),
('1000235', datetime '2018-02-13 22:30:00', 'click GczrYaxvkiPultZ'),
('1335656', datetime '2018-02-13 18:30:00', 'click MxONdLckpAFUHRs'),
('1335656', datetime '2018-02-13 19:54:00', 'click mKRPGOCiFDyzTgM'),
('2656199', datetime '2018-02-13 08:30:00', 'click CZwafHsbJOPNitL'),
('2656199', datetime '2018-02-13 09:14:00', 'click nYHJqIpjevKkToy'),
('2656199', datetime '2018-02-13 21:05:00', 'click gbAfPCwrGXvEjpl'),
('2656199', datetime '2018-02-13 21:08:00', 'click dhpZyWMuGjBOTJP'),
('2656199', datetime '2018-02-13 22:29:00', 'click bAsxnUdDhvfqaBr'),
('2656199', datetime '2018-02-13 22:30:00', 'click XlhZdLaOocQRmrY'),
('4356142', datetime '2018-02-13 18:30:00', 'click DYqShmGblOWKier'),
('4356142', datetime '2018-02-13 19:54:00', 'click DYqShmGblOWKier'),
('8881237', datetime '2018-02-13 00:30:00', 'click MpkvilgWSmhUuPn'),
('8881237', datetime '2018-02-13 06:14:00', 'click OkTYNUHMqZzLDyL'),
('8881237', datetime '2018-02-13 10:30:00', 'click OkTYNUHMqZzLDyL'),
('9890100', datetime '2018-02-13 16:01:00', 'click vOTQfBFjcgXisYU'),
('9890100', datetime '2018-02-13 16:20:00', 'click WxaLgOCcVEvhiFJ')
;

```

iii. Use a UDJ function in SQL.

```

SELECT r.user_id, from_unixtime(time/1000) as time, content FROM (
SELECT user_id, time as time, pay_info FROM payment
) p JOIN (
SELECT user_id, time as time, content FROM user_client_log
) u
ON p.user_id = u.user_id
USING pay_user_log_merge_join(p.time, p.pay_info, u.time, u.content)
r
AS (user_id, time, content);

```

The parameters in the USING clause are described as follows:

- `pay_user_log_merge_join` : the name of the UDJ function in SQL.
- `(p.time, p.pay_info, u.time, u.content)` : the columns used in UDJ.
- `r` : the alias of the result returned by the UDJ function. You can reference this alias in other SQL statements.
- `(user_id, time, content)` : the columns returned by the UDJ function.

The statement output is as follows:

```

+-----+-----+-----+
| user_id | time      | content |
+-----+-----+-----+
| 1000235 | 2018-02-13 00:25:36 | click FNOXAibRjklQPB |
| 1000235 | 2018-02-13 22:30:00 | click GczrYaxvkiPultZ |
| 1335656 | 2018-02-13 18:30:00 | click MxONdLckpAFUHRs, pay PEqMSHyktn |
| 1335656 | 2018-02-13 19:54:00 | click mKRPGOCiFDyzTgM, pay PEqMSHyktn |
| 2656199 | 2018-02-13 08:30:00 | click CZwafHsbjOPNitL, pay pYvotuLDIT |
| 2656199 | 2018-02-13 09:14:00 | click nYHJqlpjevkKToy, pay pYvotuLDIT |
| 2656199 | 2018-02-13 21:05:00 | click gbAfPCwrGXvEjpl, pay PEqMSHyktn |
| 2656199 | 2018-02-13 21:08:00 | click dhpZyWMuGjBOTJP, pay PEqMSHyktn |
| 2656199 | 2018-02-13 22:29:00 | click bAsxnUdDhvfqaBr, pay gZhvdYSOQb |
| 2656199 | 2018-02-13 22:30:00 | click XlhZdLaOocQRmrY, pay gZhvdYSOQb |
| 4356142 | 2018-02-13 18:30:00 | click DYqShmGbloWKier |
| 4356142 | 2018-02-13 19:54:00 | click DYqShmGbloWKier |
| 8881237 | 2018-02-13 00:30:00 | click MpkvilgWSmhUuPn, pay pYvotuLDIT |
| 8881237 | 2018-02-13 06:14:00 | click OkTYNUHMqZzLDyL, pay pYvotuLDIT |
| 8881237 | 2018-02-13 10:30:00 | click OkTYNUHMqZzLDyL, pay KBuMzRpsko |
| 9890100 | 2018-02-13 16:01:00 | click vOTQfBFjcgXisYU, pay gZhvdYSOQb |
| 9890100 | 2018-02-13 16:20:00 | click WxaLgOCcVEvhiFJ, pay MxONdLckwa |
+-----+-----+-----+

```

Pre-sorting of UDJ

If you want to find the payment record that has the closest time to a specific record with the same user_id, use an iterator to traverse all the records in the payment table. To perform this task, you must load all payment records with the same user_id to an ArrayList. This method can be used if the number of payment records is small. If a large number of payment records are generated, you must find another method to load the data due to the limits on memory size. To address this issue, you can use the SORT BY clause.

If the number of payment records is too large to be stored in the memory and all the data in the table has been sorted by time, you only need to compare the first element in these two lists.

This method uses the SORT BY clause to pre-sort the data. A maximum of only three records can be cached to achieve this result. UDJ in Java:

```
@Override
public void join(Record key, Iterator<Record> left, Iterator<Record> right, Yieldable<Record> output) {
    outputRecord.setString(0, key.getString(0));
    if (! right.hasNext()) {
        return;
    } else if (! left.hasNext()) {
        while (right.hasNext()) {
            Record logRecord = right.next();
            outputRecord.setBigint(1, logRecord.getDatetime(0).getTime());
            outputRecord.setString(2, logRecord.getString(1));
            output.yield(outputRecord);
        }
        return;
    }
    long prevDelta = Long.MAX_VALUE;
    Record logRecord = right.next();
    Record payRecord = left.next();
    Record lastPayRecord = payRecord.clone();
    while (true) {
        long delta = logRecord.getDatetime(0).getTime() - payRecord.getDatetime(0).getTime();
        if (left.hasNext() && delta > 0) {
            // The time delta between the two records decreases and the operation can continue.
            // Explore the group on the left to try to obtain a smaller time delta.
            lastPayRecord = payRecord.clone();
            prevDelta = delta;
            payRecord = left.next();
        } else {
            // The minimum time delta point is reached. Check the final salary record in the payment table.
            // Generate the merged result and prepare to process the next record.
            // The group on the right.
```

```
Record nearestPay = Math.abs(delta) < prevDelta ? payRecord : lastPayRecord;
outputRecord.setBigint(1, logRecord.getDatetime(0).getTime());
String mergedString = mergeLog(nearestPay.getString(1), logRecord.getString(1));
outputRecord.setString(2, mergedString);
output.yield(outputRecord);
if (right.hasNext()) {
    logRecord = right.next();
    prevDelta = Math.abs(
        logRecord.getDatetime(0).getTime() - lastPayRecord.getDatetime(0).getTime()
    );
} else {
    break;
}
}
```

3.19. Appendix

3.19.1. Escape character

In MaxCompute SQL, the backslash character (\) is an escape character that invokes an alternative representation on the following characters in a character sequence or indicates a literal interpretation of these characters.

escape character

If the backslash character (\) in a string constant is followed by three valid octal digits in the range from 001 to 177, the combination of the backslash and the three digits is interpreted into an ASCII character.

Escape sequence

MaxCompute SQL uses the backslash character (\) as an escape character for the escape sequences in the following table.

Escape sequence	Represented character
\b	backspace
\t	tab
\n	newline
\r	carriage-return
\'	Single quotation mark

Escape sequence	Represented character
\"	Double quotation mark
\\	Backslash
\;	Semicolon
\Z	control-Z
\0 or \00	Terminator

Example

- Use an escape sequence to represent a string constant.

In MaxCompute SQL, you can represent a string constant by using single quotation marks or double quotation marks. A string constant that contains a double quotation mark can be enclosed in a pair of single quotation marks. A string constant that contains a single quotation mark can be enclosed in a pair of double quotation marks. You can also use the backslash (\) as an escape character to represent a string constant. The following example shows two ways to represent a string constant:

```
"I'm a happy manong."
'I\'m a happy manong.'
```

- Use an escape sequence to represent a special character. For example, run the following command:

```
select length('a\tb') from dual;
```

The preceding command returns 3, which indicates that the string contains three characters and that `\t` is treated as one character.

```
select 'a\ab',length('a\ab') from dual;
```

The preceding command returns aab, 3, which indicates that `\a` is treated as letter a.

3.19.2. LIKE usage

In LIKE matching, '%' indicates matching any multiple characters. The '_' indicates matching a single character. To match '%' or '_' itself, you must escape it. The '\\%' matches the character '%' and '_' matches the character '_'.

```
'abcd' like 'ab%' -- true
'abcd' like 'ab_' -- false
'ab_cde' like 'ab\\_c%'; -- true
```

 **Notice** MaxCompute SQL only supports the UTF-8 character set. If the data is encoded in another format, it is possible that the calculation result is not correct.

3.19.3. Regular expression

The regular expressions in MaxCompute SQL use the PCRE standard, matched by characters. The meta character to be supported is as follows:

Metacharacter	Description
^	Top of line (TOL)
\$	End of line
.	Any character
*	Matches for zero or multiple times
+	Matches for once or multiple times
?	Matches for zero time or once
?	Matches modifier. When this character follows any other constraints (*, +, ? {n}, {n, {n, m}}), the match mode is non greedy. Non greedy mode matches strings as little as possible, while the default greedy mode matches strings as more as possible.
A B	A or B
(abc)*	Matches 'abc' for zero or multiple times
{n} or {m, n}	Matching times
[ab]	Matches any character in the brackets. In the example, it is to match a or b.
[a-d]	Matches any character in a, b, c, and d.
[^ab]	^ indicates 'non', to match any character which is not a and b.
[::]	See POSIX character group in next table.
\	Escape character
\n	N is a digit from 1 to 9 and is backward referenced.
\d	digits
\D	Non-number

POSIX character group:

POSIX Character Group	Description	Range
<code>[:alnum:]</code>	letter and digit characters	<code>[a-zA-Z0-9]</code>
<code>[:alpha:]</code>	letter	<code>[a-zA-Z]</code>
<code>[:ascii:]</code>	ASCII character	<code>[\x00-\x7F]</code>
<code>[:blank:]</code>	Space character and tabs	<code>[\t]</code>
<code>[:cntrl:]</code>	Control character	<code>[\x00-\x1F\x7F]</code>
<code>[:digit:]</code>	Digit character	<code>[0-9]</code>
<code>[:graph:]</code>	Characters except white space characters	<code>[\x21-\x7E]</code>
<code>[:lower:]</code>	Lowercase characters	<code>[a-z]</code>
<code>[:print:]</code>	<code>[:graph:]</code> and white space characters	<code>[\x20-\x7E]</code>
<code>[:punct:]</code>	punctuation	<code>[]["' #&%&'()*+,-./:;<=>? @^_`{ }~]</code>
<code>[:space:]</code>	White space characters	<code>[\t\r\n\v\f]</code>
<code>[:upper:]</code>	Uppercase characters	<code>[A-Z]</code>
<code>[:xdigit:]</code>	hexadecimal character	<code>[A-Fa-f0-9]</code>

Because the system uses a backslash () as an escape character, all “\” which appear in the regular expression pattern perform two escapes. For example, the regular expression needs to match the string “a+b”. The “+” is a special character in regular expressions and must be expressed by escape. The expression in a regular engine is “a\\+b”, because the system needs to explain a layer of escape, the expression which can match this string is “a\\\\+b”.

Suppose that the table test_dual is:

```
select 'a+b' rlike 'a\\\\+b' from test_dual;
```

```
|_c1|
```

```
|true|
```

In extreme cases, to match the character “\”, because “\” is a special character in a regular engine, it needs to be expressed by “\\”, while the system does an escape for it again, it is written as “\\\\”.

```
select 'a\\b', 'a\\b' rlike 'a\\\\b' from test_dual;
```

```
| _c0 | _c1 |
```

```
| a\b | true |
```

 **Note** To write `a\\b` in MaxCompute SQL, and the output result is `a\b`.

If TAB exists in a string, when the system reads these two characters `\t`, they are already saved as one character by the system. Therefore, in regular expression, it is a general character.

```
select 'a\tb', 'a\tb' rlike 'a\tb' from test_dual;| _c0 | _c1 |
```

```
| a b | true |
```

3.19.4. Reserved words and keywords

This document shows all reserved words in MaxCompute SQL.

Notice

- These cannot be used to name a table, column, or partition, otherwise an error occurs.
- Reserved words are not case sensitive.

```

% & && ( ) * +
- . / ; < <= <>
ADD AFTER ALL
ALTER ANALYZE AND ARCHIVE ARRAY AS ASC
BEFORE BETWEEN BIGINT BINARY BLOB BOOLEAN BOTH DECIMAL
BUCKET BUCKETS BY CASCADE CASE CAST CFILE
CHANGE CLUSTER CLUSTERED CLUSTERSTATUS COLLECTION COLUMN COLUMNS
COMMENT COMPUTE CONCATENATE CONTINUE CREATE CROSS CURRENT
CURSOR DATA DATABASE DATABASES DATE DATETIME DBPROPERTIES
DEFERRED DELETE DELIMITED DESC DESCRIBE DIRECTORY DISABLE
DISTINCT DISTRIBUTE DOUBLE DROP ELSE ENABLE END
ESCAPED EXCLUSIVE EXISTS EXPLAIN EXPORT EXTENDED EXTERNAL
FALSE FETCH FIELDS FILEFORMAT FIRST FLOAT FOLLOWING
FORMAT FORMATTED FROM FULL FUNCTION FUNCTIONS GRANT
GROUP HAVING HOLD_DDLTIME IDXPROPERTIES IF IMPORT IN
INDEX INDEXES INPATH INPUTDRIVER INPUTFORMAT INSERT INT
INTERSECT INTO IS ITEMS JOIN KEYS LATERAL
LEFT LIFECYCLE LIKE LIMIT LINES LOAD LOCAL
LOCATION LOCK LOCKS LONG MAP MAPJOIN MATERIALIZED
MINUS MSCK NOT NO_DROP NULL OF OFFLINE OFFSET
ON OPTION OR ORDER OUT OUTER OUTPUTDRIVER
OUTPUTFORMAT OVER OVERWRITE PARTITION PARTITIONED PARTITIONPROPERTIES PARTITIONS
PERCENT PLUS PRECEDING PRESERVE PROCEDURE PURGE RANGE
RCFILE READ READONLY READS REBUILD RECORDREADER RECORDWRITER
REDUCE REGEXP RENAME REPAIR REPLACE RESTRICT REVOKE
RIGHT RLIKE ROW ROWS SCHEMA SCHEMAS SELECT
SEMI SEQUENCEFILE SERDE SERDEPROPERTIES SET SHARED SHOW
SHOW_DATABASE SMALLINT SORT SORTED SSL STATISTICS STATUS STORED
STREAMTABLE STRING STRUCT TABLE TABLES TABLESAMPLE TBLPROPERTIES
TEMPORARY TERMINATED TEXTFILE THEN TIMESTAMP TINYINT TO
TOUCH TRANSFORM TRIGGER TRUE TYPE UNARCHIVE UNBOUNDED UNDO
UNION UNIONTYPE UNIQUEJOIN UNLOCK UNSIGNED UPDATE USE
USING UTC UTC_TIMESTAMP VIEW WHEN WHERE WHILE DIV

```

3.19.5. Hive data type mapping table

The data type mapping table for MaxCompute and hive is as follows:

Hive Data Type	MaxCompute Data Type
BOOLEAN	Boolean

Hive Data Type	MaxCompute Data Type
TINYINT	Tinyint
SMALLINT	Smallint
INT	Int
BIGINT	Bigint
FLOAT	Float
DOUBLE	Double
Decimal	Decimal
String	String
Varchar	Varchar
Char	String
BINARY	Binary
Timestamp	Timestamp
Date	Datetime
ARRAY	Array
Map <key, value>	MAP
STRUCT	STRUCT
Union	This feature is not supported.

4.MapReduce

4.1. Summary

4.1.1. MapReduce

This topic describes the MapReduce programming interface supported by MaxCompute and its limitations.

MaxCompute provides three versions of MapReduce programming interface:

- **MaxCompute MapReduce**: Native interface for MaxCompute, which is faster than other interfaces. It is more convenient to develop a program without exposing file system.
- **MR2 (Extended MapReduce)**: The extension to MaxCompute, which supports more complex job scheduling logic. MapReduce is implemented in the same way as the MaxCompute native interface.
- **Hadoop compatible version**: Highly compatible with **Hadoop MapReduce**, but not compatible with MaxCompute native interface and MR2.

The preceding three versions are basically the same in the **Basic concepts**, **Job submission**, **Input and output**, and **Resource**, and the only difference is the Java SDK. This article introduces the principle of MapReduce. For more detailed description of MapReduce, see **Hadoop MapReduce Course**.

 **Note** You are not yet able to read or write data from the external tables through MapReduce.

Scenarios

MapReduce was originally proposed by Google as a distributed data processing model and is now widely applied in multiple business scenarios. The following are the examples:

- **Search**: web crawl, flip index, PageRank.
- **Web access log analytics**:
 - Analyze and mine the web access, shopping behavior characteristics to achieve personalized recommendation.
 - Analyze user's access behavior.
- **Statistics and analysis for the text**:
 - The Wordcount and TFIDF analysis of Mo Yan novels.
 - Reference analysis and statistics of academic papers and patent documents.
 - Wikipedia data analysis, and so on.
- **Massive Data Mining**: Unstructured data, spatial and temporal data, image data mining.
- **Machine Learning**: Supervised learning, unsupervised learning, classification algorithm such as decision tree and SVM.
- **Natural Language Processing**:
 - Training and forecasting based on big data.

- Based on the corpus to construct the current matrix of words, frequent itemset data mining, repeated document detection and so on.
- Advertisement recommendations: User-click (CTR) and purchase behavior (CVR) forecasts.

Processing data process

The processing data process of MapReduce is divided into two stages: Map and Reduce. Map must be executed first, and then Reduce. The processing logic of Map and Reduce is defined by the user, but must comply with the MapReduce framework protocol. The process is as follows:

1. Before executing Map, the input data must be sliced, that is, input data is divided into blocks of equal size. Each block is processed as the input of a single Map Worker, so that multiple Map Workers can work simultaneously.
2. After the slice is split, multiple Map Worker can work together. Each Map Worker performs computing after reading the data and output the result to Reduce. Because Map Worker outputs the data, it must specify a key for each output record. The value of this Key determines which Reduce Worker the data has been sent to. The relationship between key value and Reduce Worker is an any-to-one relationship. Data with the same key is sent to the same Reduce Worker, and a single Reduce Worker may receive data of multiple key values.
3. Before Reduce stage, MapReduce framework sorts the data according to their Key values, and make sure data with same Key value is grouped together. If a user specifies Combiner, the framework calls Combiner to aggregate the same key data. The user must define the logic of Combiner. Compared to the classical MapReduce framework, the input parameter and output parameter of Combiner must be consistent with the Reduce in MaxCompute. This processing is generally called as Shuffle.
4. At Reduce stage, data with the same key is shuffled to the same Reduce Worker. A Reduce Worker receives data from multiple Map Workers. Each Reduce Worker executes Reduce operation for multiple records of the same key. Then these multiple records become a value through Reduce processing.

 **Note** A brief introduction to the MapReduce framework is mentioned in the preceding process. For more information, see relevant documents.

The following example uses WordCount to explain the stages of MaxCompute MapReduce.

Assume that a text named 'a.txt', where each row is indicated by a number, and the frequency of appearance of each number must be counted. The number in the text is called as 'Word' and the number appearance occurrence is called as 'Count'. To complete this function through MaxCompute MapReduce, the following figure illustrates the required steps:

□

Procedure:

1. First, text is sliced and the data in each slice is entered into a single Map Worker.
2. Map processes the input. Once Map gets a number, it sets the Count as 1. Then, output <Word,Count> queues sequence is followed. Take 'Word' as the Key of output data.
3. In the initial actions of Shuffle stage, the output of each Map Worker is sorted according to Key value (value of Word). The Combine operation is executed after sorting to accumulate the Count of same Key value (Word value) and constitute a new <Word,Count> queue. This process is called as the combiner sorting.

4. In the later actions of Shuffle, data is transmitted to Reduce. Reduce Worker sorts the data based on the Key value again after receiving the data.
5. At the time of processing data, each Reduce Worker adopts that same logic as that of a Combiner by accumulating Count with the same Key value (Word value) to get the output.
6. Result.

 **Note** Because the data in MaxCompute is stored in tables, the input and output of MaxCompute MapReduce can only be a table. User-defined output is not allowed and the corresponding file system interface is not provided.

Limits

For the limits of MapReduce, see [MR limits](#).

For the limits of running MapReduce locally, see [Difference between running locally and running distributed environments](#).

4.1.2. Extended MapReduce model

The extended MapReduce model (MR2) that is provided by MaxCompute uses optimized scheduling and I/O models to reduce unnecessary I/O operations during job running.

extended MapReduce model MR2

In the extended MapReduce model, Map and Reduce functions are written in the same way as MaxCompute. The difference lies in how the jobs run. For more information, see [Pipeline samples](#).

Background information

A MapReduce model consists of multiple MapReduce jobs. In a traditional MapReduce model, the output of each MapReduce job must be written to a disk in a distributed file system such as HDFS or to a MaxCompute table. However, subsequent Map tasks may only need to read the outputs once to prepare for the Shuffle stage. This mechanism results in redundant I/O operations.

The computing and scheduling logic of MaxCompute supports more complex programming models. A Map operation can be succeeded by any number of consecutive Reduce operations without the need for a Map operation in between, such as Map-Reduce-Reduce.

Comparison with Hadoop ChainMapper and ChainReducer

Similarly, Hadoop ChainMapper and ChainReducer also support serialized Map or Reduce operations. However, they are essentially different from the extended MapReduce model.

Hadoop ChainMapper and ChainReducer are based on the traditional MapReduce model. They support only Map operations after the original Map or Reduce operation. One benefit is that you can reuse the Mapper business logic to split a Map or Reduce operation into multiple Mapper stages. However, this does not modify the scheduling or I/O models.

4.1.3. Open-source MapReduce

This article introduces the application background of open-source MapReduce and the basic usage of HadoopMR plug-in.

MaxCompute offers a set of native MapReduce programming models and interfaces. The inputs and outputs for these interfaces are MaxCompute tables, and the data is organized to be processed in the record format.

However, MaxCompute APIs differ significantly from APIs for the Hadoop framework. Previously, to migrate your Hadoop MapReduce jobs to MaxCompute, firstly, you were needed to rewrite the MapReduce code, compile, and debug the code using MaxCompute APIs, compress the final code into a JAR package, and finally upload the package to the MaxCompute platform. This process is tedious and requires a lot of development and testing efforts. If you are not required to modify the original Hadoop MapReduce code partially, running it in MaxCompute console is the best solution.

Now, the MaxCompute platform provides a plug-in that allows you to adapt Hadoop MapReduce code to MaxCompute MapReduce specifications. MaxCompute offers a degree of flexibility regarding binary-level compatibility for Hadoop MapReduce jobs. It means that, without modifying the code, you can specify configurations to directly run original Hadoop MapReduce Jar packages on MaxCompute. Download the development plug-in to get **started**. This plug-in is currently in the testing stage, therefore, does not support custom comparators or key types.

In the following example, a WordCount program is used to introduce the basic usage of the plug-in.

Download the HadoopMR Plug-in

Click [Hadoop MapReduce](#) to download the plug-in named hadoop2openmr-1.0.jar.

 **Note** This Jar package contains the dependencies with Hadoop 2.7.2. Do not include Hadoop dependencies in the Jar packages of your jobs to avoid version conflicts.

Prepare a Jar package

Compile and export the WordCount JAR package named wordcount_test.jar. The WordCount program source code is as follows:

```
package com.aliyun.odps.mapred.example.hadoop;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import java.util.StringTokenizer;
public class WordCount {
    public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{
        private final static IntWritable one = new IntWritable(1);
```

```

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context
        ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }
}

public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values,
        Context context
    ) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

Prepare the test data

1. Create input and output tables.

```
create table if not exists wc_in(line string);  
create table if not exists wc_out(key string, cnt bigint);
```

2. Run Tunnel to import data to the input table.

The data in the data.txt file to be imported is as follows:

```
hello maxcompute  
hello mapreduce
```

Use the `Tunnel command` on the MaxCompute console to import data from data.txt to wc_in.

```
tunnel upload data.txt wc_in;
```

Configure the mapping between the table and the HDFS file path

The configuration file is wordcount-table-res.conf:

```
{
  "file:/foo": {
    "resolver": {
      "resolver": "com.aliyun.odps.mapred.hadoop2openmr.resolver.TextFileResolver",
      "properties": {
        "text.resolver.columns.combine.enable": "true",
        "text.resolver.seperator": "\t"
      }
    },
    "tableInfos": [
      {
        "tblName": "wc_in",
        "partSpec": {},
        "label": "__default__"
      }
    ],
    "matchMode": "exact"
  },
  "file:/bar": {
    "resolver": {
      "resolver": "com.aliyun.odps.mapred.hadoop2openmr.resolver.BinaryFileResolver",
      "properties": {
        "binary.resolver.input.key.class" : "org.apache.hadoop.io.Text",
        "binary.resolver.input.value.class" : "org.apache.hadoop.io.LongWritable"
      }
    },
    "tableInfos": [
      {
        "tblName": "wc_out",
        "partSpec": {},
        "label": "__default__"
      }
    ],
    "matchMode": "fuzzy"
  }
}
```

Parameters

The configuration is a JSON file that describes the mapping relationships between HDFS files and the MaxCompute tables. Generally, you must configure both the input and output. One HDFS path corresponds to one Resolver, tableInfos, and matchMode.

- **resolver:** Specifies the method of processing file data. Currently, you can choose from two built-in Resolvers: `com.aliyun.odps.mapred.hadoop2openmr.resolver.TextFileResolver` and `com.aliyun.odps.mapred.hadoop2openmr.resolver.BinaryFileResolver`. In addition to specifying the Resolver name, configure some properties about data parsing for the Resolver.
 - **TextFileResolver:** Regards an input or output as plain text if the data is of plain text type. When configuring an input Resolver, configure such properties as `text.resolver.columns.combine.enable` and `text.resolver.seperator`. When `text.resolver.columns.combine.enable` is set to true, all the columns in the input table are combined into a single string based on the delimiter specified by `text.resolver.seperator`. Otherwise, the first two columns in the input table are used as the key and value.
 - **BinaryFileResolver:** Converts binary data into a type that is supported by MaxCompute, for example, Bigint, Boolean, and Double. When configuring an output Resolver, configure the properties `binary.resolver.input.key.class` and `binary.resolver.input.value.class`, which define the key and value types of the intermediate result, respectively.
- **tableInfos:** Specifies the MaxCompute table that corresponds to HDFS. Currently, only the `tblName` parameter (table name) is configurable. The `partSpec` and `label` parameters must be the same as the values set for the parameters in this example.
- **matchMode:** Specifies the path matching mode. The exact mode indicates exact matching, and the fuzzy mode indicates fuzzy matching. Use a regular expression in fuzzy mode to match the HDFS input path.

Job Submission

Use the MaxCompute command line tool `odpscmd` to submit jobs. For the installation and configuration of MaxCompute command line tool, see the [Console](#). In `odpscmd`, run the following command:

```
jar -DODPS_HADOOPMR_TABLE_RES_CONF=./wordcount-table-res.conf -classpath hadoop2openmr-1.0.jar,wordcount_test.jar com.aliyun.odps.mapred.example.hadoop.WordCount /foo/bar;
```

 Note

- wordcount-table-res.conf is a map with /foo/bar configured.
- wordcount_test.jar is your Jar package of Hadoop MapReduce.
- com.aliyun.odps.mapred.example.hadoop.WordCount is the class name of job to be run.
- /foo/bar refers to the path on HDFS, which is mapped to wc_in and wc_out in the JSON configuration file.
- With the mapping relation configured, manually import the Hadoop HDFS input file to wc_in for MR calculations by using data integration functions of DataX or DataWorks, and manually export the result wc_out to your HDFS output directory(/bar).
- In the preceding output, assume that hadoop2openmr-1.0.jar, wordcount_test.jar, and wordcount-table-res.conf are stored in the current directory of odpscmd. If an error occurs, make the relevant changes when specifying the configuration and -classpath.

The running process is as follows:

□

After running the job, check the results table wc_out to verify whether a job is complete:

□

4.2. Function Introduction

4.2.1. Basic concepts

Map/Reduce

Map and Reduce support corresponding Map/Reduce methods, setup methods, and cleanup methods. The setup method is called before the Map/Reduce method, and each worker calls it only once.

The cleanup method is called after the map/reduce method, and each worker calls it only once.

For a detailed example, see [Program examples](#).

Sort/Group

Some columns in output key records can be taken as sort columns, but user-defined comparator is not supported. You can select several columns from the sort column as Group columns, but the user-defined Group comparator is not supported. Sort columns are used to sort your data while Group columns are used for a Secondary Sort.

For more information, see [SecondarySort Example](#).

Partition

Supports setting the partition column and customized partitioner. Partition columns have a higher priority than customized partitioners.

According to Hash logic, the partitioner distributes the output data on the Map terminal to different Reduce Workers.

Combiner

Combines adjacent records in the Shuffle stage. You can choose whether to use Combiner according to different business logic.

Combiner helps to optimize the MapReduce computing framework and the logic of Combiner is generally similar to Reduce. After Map outputs the data, the framework performs a local combiner operation for the data which has the same key value on the Map terminal.

For more information, see [WordCount code examples](#).

4.2.2. Commands

The MaxCompute console provides a JAR command to run MapReduce job. The detailed syntax is shown as follows:

Usage:

```
jar [<GENERIC_OPTIONS>] <MAIN_CLASS> [ARGS];
```

-conf <configuration_file> Specify an application configuration file

-resources <resource_name_list> file\table resources used in mapper or reducer, separate by comma

-classpath <local_file_list> classpaths used to run mainClass

-D <name>=<value> Property value pair, which will be used to run mainClass

-l Run job in local mode

For example:

```
jar -conf /home/admin/myconf -resources a.txt,example.jar -classpath ../lib/example.jar:../other_lib.jar -Djava.library.path=./native -Xmx512M mycompany.WordCount -m 10 -r 10 in out;
```

<GENERIC_OPTIONS> includes the following parameters (optional parameters):

- -conf <configuration file>: Specify an JobConf configuration file.
- -resources <resource_name_list>: Indicates the resource statement used in MapReduce running time. Generally, the resource name in which Map/Reduce function is included must be specified in 'resource_name_list'.

Note

If the user has read other MaxCompute resources in the Map/Reduce function, then these resource names also must be added in 'source_name_list'.

Multiple resources are separated by commas (,). If you must use span project resources, then add the prefix `PROJECT/resources/`, for example: `-resources otherproject/resources/resourcefile`.

For more information about how to read the resource in the Map/Reduce function, see [Use Resource Example](#).

- -classpath <local_file_list>: the classpath used to specify the local JAR package of 'main' class (include relative paths and absolute paths).

Package names are separated using system default file delimiters. Generally, the delimiter is a semicolon (;) in a Windows system and a comma (,) in a Linux system.

 **Note** In most cases, users generally write the main class and Map/Reduce function in a package, such as [WordCount Code Example](#). This means that, in the running period of the example program, mapreduce-examples.jar appears in '-resources' parameter and '-classpath' parameter, however, '-resources' references the Map/Reduce function, and runs in a distributed environment, while '-classpath' references 'Main' class, and runs locally. The specified path of the JAR package is also a local path.

- -D <prop_name>=<prop_value> : Multiple Java properties of <mainClass> in a local mode can be defined.
- -l: run MapReduce job in local mode, mainly used for program debugging.

User can specify the configuration file 'JobConf' by option '-conf'. This file can modify the JobConf settings in the SDK.

An example of a configuration file 'JobConf' is as follows:

```
<configuration>
  <property>
    <name>import.filename</name>
    <value>resource.txt</value>
  </property>
</configuration>
```

In the preceding example, the variable 'import.filename' is defined and its value is 'resource.txt'.

User can get this variable value through the JobConf interface in the MapReduce program. Alternatively, users can also get the value through the JobConf interface in the SDK. For a detailed example, see [Use Resource Example](#).

Example:

```
add jar data\mapreduce-examples.jar;
jar -resources mapreduce-examples.jar -classpath mapreduce-examples.jar
    org.alidata.odps.mr.examples.WordCount wc_in wc_out;
add file data/src.txt;
add jar data\mapreduce-examples.jar;
jar -resources src.txt,mapreduce-examples.jar -classpath data\mapreduce-examples.jar
    org.alidata.odps.mr.examples.WordCount wc_in wc_out;
add file data/a.txt;
add table wc_in as test_table;
add jar data/work.jar;
jar -conf odps-mapred.xml -resources a.txt,test_table,work.jar
    -classpath data/work.jar:otherlib.jar
    -D import.filename=resource.txt org.alidata.odps.mr.examples.WordCount args;
```

4.2.3. Input and output

This topic describes the input and output of MapReduce jobs in MaxCompute.

input and output

- The input and output of MapReduce jobs in MaxCompute support built-in data types of MaxCompute, including BIGINT, DOUBLE, STRING, DATETIME, and BOOLEAN. User-defined data types are not supported.
- MapReduce supports input data from multiple tables with different schemas. You can use the map function to obtain the table information corresponding to the current record.
- MapReduce supports null values as input data, but does not support views as input data.
- A reduce job can write outputs to different tables or different partitions of a table. The target tables can have different schemas. Different outputs are distinguished by labels. By default, no label is added to the default output. MapReduce does not support a function without output returned.

For more information about input and output examples, see [MultipleInOut](#).

4.2.4. Resources

You can learn more about MaxCompute resources in the Map/Reduce section. Any Worker of Map/Reduce can load resources to the memory for you to apply the code for further use.

For more information, see [Use resource example](#).

4.2.5. Local run

Basic stages introduction

Local run prerequisite: By setting -local parameter in the jar command, user can simulate MapReduce running process on the local to initiate local debugging.

At local operation time: The client downloads required meta information of input tables, resources, and meta information of output tables from MaxCompute, and saves them into a local directory named 'warehouse'.

After running the program: The calculation result is output into a file in the 'warehouse'. If the input table and referenced resources have been downloaded in the local warehouse directory, the data and files in 'warehouse' directory are referenced directly during the next run time, and the downloading process does not need to be repeated.

Difference between running locally and running distributed environments

In the local operation course, multiple Map and Reduce workers are yet to start data processing. But these workers do not run concurrently and run serially.

The distinguishing points between the simulation process and real distributed operation are as follows:

- A limit on the row number of input table exists. Currently, up to 100 rows of data can be downloaded.
- Usage of resources: In a distributed environment, MaxCompute limits the size of the referenced resource. For more information, see [Application Restriction](#). Note that in the local running environment, the resource size has no limits.
- Security limits: MaxCompute, MapReduce, and UDF program running in a distributed environment are limited by [Java Sandbox](#). Note that in local operations this limit is not applicable.

Example:

A local operation example is as follows:

```
odps:my_project> jar -l com.aliyun.odps.mapred.example.WordCount wc_in wc_out
Summary:
counters: 10
  map-reduce framework
    combine_input_groups=2
    combine_output_records=2
    map_input_bytes=4
    map_input_records=1
    map_output_records=2
    map_output_[wc_out]_bytes=0
    map_output_[wc_out]_records=0
    reduce_input_groups=2
    reduce_output_[wc_out]_bytes=8
    reduce_output_[wc_out]_records=2
OK
```

For a detailed WordCount example, see [WordCount Code example](#).

If a user runs local debugging command for the first time, a path named 'warehouse' appears in the current path after the command is executed successfully. The directory structure of warehouse is as follows:

```
<warehouse>
|__my_project(project directory)
|  |__<_tables__>
|  |__wc_in(table directory)
|  |  |__data(file)
|  |  |
|  |  |__<_schema__> (file)
|  |  |__wc_out(table data directory)
|  |  |__data(file)
|  |  |
|  |  |__<_schema__> (file)
|  |  |
|  |  |__<_resources__>
|  |  |
|  |  |__table_resource_name (table resource)
|  |  |__<_ref__>
|  |  |
|  |  |__file_resource_name (file resource)
```

- The same level directory of myproject indicates the project. 'wc_in' and 'wc_out' indicate tables. The table files read by user in JAR command is downloaded into this directory.
- The contents in <_schema__> indicate table meta information. The format is defined as follows:

```
project=local_project_name
table=local_table_name
columns=col1_name:col1_type,col2_name:col2_type
partitions=p1:STRING,p2:BIGINT
```

Columns and column types are separated by colons (:), and columns are separated by commas (,). Corresponding to <_schema__> file, the Project name and Table name must be declared, such as `project_name.table_name`, and separated by a comma (,) and column definition. `project_name.table_name,col1_name:col1_type,col2_name:col2_type,.....`

- The file 'data;' indicates the table data. The column quantity and corresponding data must comply with the definition in `_schema_`. Moreover, extra columns and missing columns are not allowed.

The content of `_schema_` in `wc_in` is as follows:

```
my_project.wc_in,key:STRING,value:STRING
```

The content of 'data' is as follows:

```
0,2
```

The client downloads the meta information of table and part of the data from MaxCompute, and save them into the two preceding files. If you run this example again, the data in the directory 'wc_in' is used directly and will not be downloaded again.

The content of '_schema_' in wc_out is as follows:

```
my_project.wc_out,key:STRING,cnt:BIGINT
```

The content of 'data' is as follows:

```
0,1
2,1
```

The client downloads the meta information of wc_out from MaxCompute and saves it to the file `_schema_`. The file 'data' is a result data file generated after the local operation.

 Note

- Users can also edit `_schema_` file and 'data' and then place these two files into the corresponding table directory.
- When running on the local, the client can detect the table directory already exists, and does not download the information of this table from MaxCompute. The table directory on the local can be a table that does not exist in MaxCompute.

4.3. Program Example

4.3.1. WordCount samples

Prerequisites

1. Prepare a Jar package of the test program. Assume the package is named `mapreduce-examples.jar`, and the local storage path is `data/resources`.

- Create tables:

```
create table wc_in (key string, value string);
create table wc_out(key string, cnt bigint);
```

- Add resources:

```
add jar data/resources/mapreduce-examples.jar -f;
```

2. Prepare tables and resources for testing the WordCount operation.
3. Run tunnel to import data.

```
tunnel upload data wc_in;
```

The contents of data file imported into the table wc_in, as follows:

```
hello,odps
```

Procedure

Run WordCount in odpscmd.

```
jar -resources mapreduce-examples.jar -classpath data/resources/mapreduce-examples.jar  
com.aliyun.odps.mapred.open.example.WordCount wc_in wc_out
```

Expected output

The content of output table wc_out is as follows:

```
+-----+-----+  
| key | cnt |  
+-----+-----+  
| hello | 1 |  
| odps | 1 |  
+-----+-----+
```

Sample code

```
package com.aliyun.odps.mapred.open.example;  
import java.io.IOException;  
import java.util.Iterator;  
import com.aliyun.odps.data.Record;  
import com.aliyun.odps.data.TableInfo;  
import com.aliyun.odps.mapred.JobClient;  
import com.aliyun.odps.mapred.MapperBase;  
import com.aliyun.odps.mapred.ReducerBase;  
import com.aliyun.odps.mapred.TaskContext;  
import com.aliyun.odps.mapred.conf.JobConf;  
import com.aliyun.odps.mapred.utils.InputUtils;  
import com.aliyun.odps.mapred.utils.OutputUtils;  
import com.aliyun.odps.mapred.utils.SchemaUtils;  
public class WordCount {  
    public static class TokenizerMapper extends MapperBase {  
        private Record word;  
        private Record one;  
        @Override  
        public void setup(TaskContext context) throws IOException {  
            word = context.createMapOutputKeyRecord();  
        }  
    }  
}
```

```

        one = context.createMapOutputValueRecord();
        one.set(new Object[] { 1L });
        System.out.println("TaskID:" + context.getTaskID().toString());
    }
    @Override
    public void map(long recordNum, Record record, TaskContext context)
        throws IOException {
        for (int i = 0; i < record.getColumnCount(); i++) {
            word.set(new Object[] { record.get(i).toString() });
            context.write(word, one);
        }
    }
}
/**
 * A combiner class that combines map output by sum them.
 */
public static class SumCombiner extends ReducerBase {
    private Record count;
    @Override
    public void setup(TaskContext context) throws IOException{
        count = context.createMapOutputValueRecord();
    }
    // Assemblyer implements the same interface as reducer, you can immediately reduce the output
    of the mapper for a reduce that is performed locally on the mapper.
    @Override
    public void reduce(Record key,Iterator<Record>values,TaskContext context)
        throws IOException {
        long c = 0;
        while(values.hasNext()) {
            Record val = values.next();
            c += (Long) val.get(0);
        }
        count.set(0, c);
        context.write(key, count);
    }
}
/**
 * A reducer class that just emits the sum of the input values.
 */
public static class SumReducer extends ReducerBase {

```

```

private Record result = null;

@Override
public void setup(TaskContext context) throws IOException{
    result = context.createOutputRecord();
}

@Override
public void reduce(Record key,Iterator<Record>values,TaskContext context)
    Throws IOException {
    Long Count = 0;
    while(values.hasNext()) {
        Record val = values.next();
        count += (Long) val.get(0);
    }
    result.set(0, key.get(0));
    result.set(1, count);
    context.write(result);
}
}

public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("Usage: WordCount <in_table> <out_table>");
        System.exit(2);
    }
    JobConf job = new JobConf();
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(SumCombiner.class);
    job.setReducerClass(SumReducer.class);

    // The schema that sets the key and value of the mapper's intermediate result, the mapper's intermediate output is also the form of a record.
    job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
    job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
    // Set input and output table information
    InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    Jobclient.runjob(job);
}
}

```

4.3.2. MapOnly samples

For MapOnly jobs, Map directly sends <Key,Value> pairs to tables on MaxCompute. You only need to specify the output table. However, you can skip specifying the Key/Value metadata to be output by Map.

Prerequisites

1. Prepare a JAR package of the test program. Assume the package is named `mapreduce-examples.jar`, the local storage path is `data\resources`.
2. Prepare tables and resources for testing the MapOnly operation.
 - Create tables:

```
create table wc_in (key string, value string);
create table wc_out(key string, cnt bigint);
```

- Add resources:

```
add jar data\resources\mapreduce-examples.jar -f;
```

3. Use the tunnel command to import the data:

```
tunnel upload data wc_in;
```

The contents of data file are imported into the “`mr_src`” table:

```
hello,odps
hello,odps
```

Procedure

Run MapOnly in `odpscmd`:

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.MapOnly wc_in wc_out map
```

Expected output

The content of output table `wc_out` is as follows:

```
+-----+-----+
| key | cnt |
+-----+-----+
| hello | 1 |
| hello | 1 |
+-----+-----+
```

Sample code

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
```

```
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.mapperbase;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.data.TableInfo;
public class MapOnly {
    public static class MapperClass extends MapperBase {
        @Override
        public void setup(TaskContext context) throws IOException{
            boolean is = context.getJobConf().getBoolean("option.mapper.setup", false);
            // The Main function sets option.mapper.setup to true in jobconf to execute the following logic.
            if (is) {
                Record result = context.createOutputRecord();
                result.set(0, "setup");
                result.set(1, 1L);
                context.write(result);
            }
        }
        @Override
        public void map(long key, Record record, TaskContext context) throws IOException {
            boolean is = context.getJobConf().getBoolean("option.mapper.map", false);
            // The Main function sets option.mapper.map to true in jobconf to execute the following logic.
            if (is) {
                Record result = context.createOutputRecord();
                result.set(0, record.get(0));
                result.set(1, 1L);
                context.write(result);
            }
        }
        @Override
        public void cleanup(TaskContext context) throws IOException {
            boolean is = context.getJobConf().getBoolean("option.mapper.cleanup", false);
            // The Main function sets option.mapper.cleanup to true in jobconf to execute the following logic
            if (is) {
                Record result = context.createOutputRecord();
                result.set(0, "cleanup");
            }
        }
    }
}
```

```

        result.set(1, 1L);
        context.write(result);
    }
}

public static void main(String[] args) throws Exception {
    if (args.length != 2 && args.length != 3) {
        System.err.println("Usage: OnlyMapper <in_table> <out_table> [setup|map|cleanup]");
        System.exit(2);
    }
    JobConf job = new JobConf();
    job.setMapperClass(MapperClass.class);
    // For maponly jobs, the number of reducers must be explicitly set to 0
    job.setNumReduceTasks(0);
    // Set table information for Input Output
    InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    if (args.length == 3) {
        String options = new String(args[2]);
        // Jobconf can set custom key, value, and getJobConf can get relevant settings in mapper through
        getJobConf of context.
        if (options.contains("setup")) {
            job.setBoolean("option.mapper.setup", true);
        }
        if (options.contains("map")) {
            job.setBoolean("option.mapper.map", true);
        }
        if (options.contains("cleanup")) {
            job.setBoolean("option.mapper.cleanup", true);
        }
    }
    Jobclient.runjob(job);
}
}

```

4.3.3. Multi-input and Output

Prerequisites

1. Prepare a Jar package of the test program. Assume the package is named `mapreduce-examples.jar`, and the local storage path is `data/resources`.

2. Prepare tables and resources for testing the multi-input and output operations.

○ Create tables:

```
create table wc_in1(key string, value string);
create table wc_in2(key string, value string);
create table mr_multiinout_out1 (key string, cnt bigint);
create table mr_multiinout_out2 (key string, cnt bigint) partitioned by (a string, b string);
alter table mr_multiinout_out2 add partition (a='1', b='1');
alter table mr_multiinout_out2 add partition (a='2', b='2');
```

○ Add resources:

```
add jar data/resources/mapreduce-examples.jar -f;
```

3. Run tunnel to import data.

```
tunnel upload data1 wc_in1;
tunnel upload data2 wc_in2;
```

The data imported into the wc_in1 table is as follows:

```
hello,odps
```

The data imported into the wc_in2 table is as follows:

```
hello,world
```

Procedure

Run MultipleInOut in odpscmd.

```
jar -resources mapreduce-examples.jar -classpath data/resources/mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.MultipleInOut wc_in1,wc_in2 mr_multiinout_out1,mr_multiinout_
out2|a=1/b=1|out1,mr_multiinout_out2|a=2/b=2|out2;
```

Expected output

The content of 'mr_multiinout_out1' is as follows:

```
+-----+-----+
| key | cnt |
+-----+-----+
| default | 1 |
+-----+-----+
```

The content of 'mr_multiinout_out2' is as follows:

```

+-----+-----+---+---+
| key | cnt | a | b |
+-----+-----+---+---+
| odps | 1 | 1 | 1 |
| world | 1 | 1 | 1 |
| out1 | 1 | 1 | 1 |
| hello | 2 | 2 | 2 |
| out2 | 1 | 2 | 2 |
+-----+-----+---+---+

```

Sample code

```

package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import java.util.LinkedHashMap;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
 * Multi input & output example.
 */
public class MultipleInOut {
    public static class TokenizerMapper extends MapperBase {
        Record word;
        Record one;
        @Override
        public void setup(TaskContext context) throws IOException{
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.set(new Object[] { 1L });
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)

```

```

    public void map(long recordnum, Record record, TaskContext context)
        Throws IOException {
        for (int i = 0; i < record.getColumnCount(); i++) {
            word.set(new Object[] { record.get(i).toString() });
            context.write(word, one);
        }
    }
}

public static class SumReducer extends ReducerBase {
    private Record result;
    private Record result1;
    private Record result2;
    @Override
    public void setup(TaskContext context) throws IOException{
        // For different outputs you need to create different records, which are distinguished by label
        result = context.createOutputRecord();
        result1 = context.createOutputRecord("out1");
        result2 = context.createOutputRecord("out2");
    }
    @Override
    public void reduce(Record key,Iterator<Record>values,TaskContext context)
        Throws IOException {
        Long Count = 0;
        while(values.hasNext()) {
            Record val = values.next();
            count += (Long) val.get(0);
        }
        long mod = count % 3;
        if (mod == 0) {
            result.set(0, key.get(0));
            result.set(1, count);
            // No label is specified. Default output is adopted.
            context.write(result);
        } else if (mod == 1) {
            result1.set(0, key.get(0));
            result1.set(1, count);
            context.write(result1, "out1");
        } else {
            result2.set(0, key.get(0));
            result2.set(1, count);
            context.write(result2, "out2");
        }
    }
}

```

```

    }
}
@Override
public void cleanup(TaskContext context) throws IOException {
    Record result = context.createOutputRecord();
    result.set(0, "default");
    result.set(1, 1L);
    context.write(result);
    Record result1 = context.createOutputRecord("out1");
    result1.set(0, "out1");
    result1.set(1, 1L);
    context.write(result1, "out1");
    Record result2 = context.createOutputRecord("out2");
    result2.set(0, "out2");
    result2.set(1, 1L);
    context.write(result2, "out2");
}
}

// Convert the partition string such as "ds = 1/pt = 2" into map form
public static LinkedHashMap<String, String> convertPartSpecToMap(
    String partSpec) {
    LinkedHashMap<String, String> map = new LinkedHashMap<String, String>();
    if (partSpec != null && ! partSpec.trim().isEmpty()) {
        String[] parts = partSpec.split("/");
        for (String part : parts) {
            String[] ss = part.split("=");
            if (ss.length != 2) {
                throw new RuntimeException("ODPS-0730001: error part spec format: "
                    + partSpec);
            }
            map.put(ss[0], ss[1]);
        }
    }
    return map;
}

public static void main(String[] args) throws Exception {
    String[] inputs = null;
    String[] outputs = null;
    if (args.length == 2) {
        inputs = args[0].split(",");
        outputs = args[1].split(",");
    }
}

```

```

    } else {
        System.err.println("MultipleInOut in... out..." );
        System.exit(1);
    }

    JobConf job = new JobConf();
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
    job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
    // Parse the user input table strings.
    for (String in : inputs) {
        String[] ss = in.split("\\\\");
        if (ss.length == 1) {
            InputUtils.addTable(TableInfo.builder().tableName(ss[0]).build(), job);
        } else if (ss.length == 2) {
            LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
            InputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(), job);
        } else {
            System.err.println("Style of input: " + in + " is not right");
            System.exit(1);
        }
    }

    // Parse the user output table strings.
    for (String out : outputs) {
        String[] ss = out.split("\\\\");
        if (ss.length == 1) {
            OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).build(), job);
        } else if (ss.length == 2) {
            LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
            OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(), job);
        } else if (ss.length == 3) {
            if (ss[1].isEmpty()) {
                LinkedHashMap<String, String> map = convertPartSpecToMap(ss[2]);
                OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map).build(), job);
            } else {
                LinkedHashMap<String, String> map = convertPartSpecToMap(ss[1]);
                OutputUtils.addTable(TableInfo.builder().tableName(ss[0]).partSpec(map)
                    .label(ss[2]).build(), job);
            }
        } else {
            System.err.println("Style of output: " + out + " is not right");
        }
    }

```



```

        System.err.println("Style of output: " + out + " is not right");
        System.exit(1);
    }
}
Jobclient.runjob(job);
}
}

```

4.3.4. Multi-task samples

Prerequisites

1. Prepare the Jar package of the test program. Assume the package is named `mapreduce-examples.jar`, and the local storage path is `data/resources`.
2. Prepare tables and resources for testing the Multijobs operation.
 - Create tables:

```

create table mr_empty (key string, value string);
create table mr_multijobs_out (value bigint);

```

- Add resources:

```

add table mr_multijobs_out as multijobs_res_table -f;
Add jar data \ resources \ mapreduce-examples.jar -f;

```

Procedure

Run Multijobs in `odpscmd`.

```

jar -resources mapreduce-examples.jar,multijobs_res_table -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Multijobs mr_multijobs_out;

```

Expected output

The output table 'mr_multijobs_out' is as follows:

```

+-----+
| value |
+-----+
| 0 |
+-----+

```

Sample code

```

package com.aliyun.odps.mapred.open.example;

```

```
package com.aliyun.odps.mapred.open.example;

import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;

/**
 * Multijobs
 *
 * Running multiple job
 *
 */
public class Multijobs {
    public static class InitMapper extends MapperBase {
        @Override
        public void setup(TaskContext context) throws IOException {
            Record record = context.createOutputRecord();
            long v = context.getJobConf().getLong("multijobs.value", 2);
            record.set(0, v);
            context.write(record);
        }
    }

    public static class DecreaseMapper extends MapperBase {
        @Override
        public void cleanup(TaskContext context) throws IOException {
            // Obtain the variable values defined by the main function from JobConf.
            long expect = context.getJobConf().getLong("multijobs.expect.value", -1);
            long v = -1;
            int count = 0;

            // Read the data in the resource table, which is the output table of the previous job
            Iterator<Record> iter = context.readResourceTable("multijobs_res_table");
            while (iter.hasNext()) {
                Record r = iter.next();
                v = (Long) r.get(0);
            }
        }
    }
}
```

```

        if (expect != v) {
            throw new IOException("expect: " + expect + ", but: " + v);
        }
        count++;
    }
    if (count != 1) {
        throw new IOException("res_table should have 1 record, but: " + count);
    }
    Record record = context.createOutputRecord();
    v--;
    record.set(0, v);
    context.write(record);
    // Sets counter, which can be obtained in the main function after the job has completed successfully
    context.getCounter("multijobs", "value").setValue(v);
}
}

public static void main(String[] args) throws Exception {
    if (args.length != 1) {
        System.err.println("Usage: TestMultijobs <table>");
        System.exit(1);
    }
    String tbl = args[0];
    long iterCount = 2;
    System.err.println("Start to run init job.");
    JobConf initJob = new JobConf();
    initJob.setLong("multijobs.value", iterCount);
    initJob.setMapperClass(InitMapper.class);
    InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), initJob);
    OutputUtils.addTable(TableInfo.builder().tableName(tbl).build(), initJob);
    initJob.setMapOutputKeySchema(SchemaUtils.fromString("key:string"));
    initJob.setMapOutputValueSchema(SchemaUtils.fromString("value:string"));
    // Maponly job needs to explicitly set reducer number to 0
    initJob.setNumReduceTasks(0);
    JobClient.runJob(initJob);
    while (true) {
        System.err.println("Start to run iter job, count: " + iterCount);
        JobConf decJob = new JobConf();
        decJob.setLong("multijobs.expect.value", iterCount);
        decJob.setMapperClass(DecreaseMapper.class);
        InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), decJob);

```

```
OutputUtils.addTable(TableInfo.builder().tableName(tbl).build(), decJob);
// Maponly job needs to explicitly set reducer number to 0
decJob.setNumReduceTasks(0);
RunningJob rJob = JobClient.runJob(decJob);
iterCount--;
// Exit the loop if the number of iterations has been reached
if (rJob.getCounters().findCounter("multijobs", "value").getValue() == 0) {
    break;
}
}
if (iterCount != 0) {
    throw new IOException("Job failed.") ;
}
}
```

4.3.5. Secondary Sort samples

Prerequisites

1. Prepare a JAR package of the test program. Assume the package is named "mapreduce-examples.jar". The local storage path is *data/resources*.
2. Prepare tables and resources for testing the SecondarySort operation.

- Create tables:

```
create table ss_in(key bigint, value bigint);
create table ss_out(key bigint, value bigint)
```

- Add resources:

```
add jar data/resources/mapreduce-examples.jar -f;
```

3. Import the data through tunnel command:

```
tunnel upload data ss_in;
```

The contents of data file imported into the table "ss_in" are as follows:

```
1,2
2,1
1,1
2,2
```

Procedure

Run SecondarySort on the odpscmd:

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.SecondarySort ss_in ss_out;
```

Expected output

The contents in the output table "ss_out" are as follows:

```
| key | value |
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |
| 2 | 2 |
```

Sample code

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.data.TableInfo;

* This is an example ODPS Map/Reduce application. It reads the input table that
* must contain two integers per record. The output is sorted by the first and
* second number and grouped on the first number.

public class SecondarySort {

    * Read two integers from each line and generate a key, value pair as ((left,
```

```
* right), right).
```

```
public static class MapClass extends MapperBase {
    private Record key;
    private Record value;
    @Override
    public void setup(TaskContext context) throws IOException {
        key = context.createMapOutputKeyRecord();
        value = context.createMapOutputValueRecord();

        @Override
        public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
            long left = 0;
            long right = 0;
            if (record.getColumnCount() > 0) {
                left = (Long) record.get(0);
                if (record.getColumnCount() > 1) {
                    right = (Long) record.get(1);

                key.set(new Object[] { (Long) left, (Long) right });
                value.set(new Object[] { (Long) right });
                context.write(key, value);
```

```
* A reducer class that just emits the sum of the input values.
```

```
public static class ReduceClass extends ReducerBase {
    private Record result = null;
    @Override
    public void setup(TaskContext context) throws IOException {
        result = context.createOutputRecord();

        @Override
        public void reduce(Record key, Iterator<Record> values, TaskContext context)
            throws IOException {
            result.set(0, key.get(0));
            while (values.hasNext()) {
                Record value = values.next();
```

```

    result.set(1, value.get(0));
    context.write(result);

}

public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("Usage: secondarysort <in> <out>");
        System.exit(2);
    }

    JobConf job = new JobConf();
    job.setMapperClass(MapClass.class);
    job.setReducerClass(ReduceClass.class);
    // set multiple columns to key
    // compare first and second parts of the pair
    job.setOutputKeySortColumns(new String[] { "i1", "i2" });
    // partition based on the first part of the pair
    job.setPartitionColumns(new String[] { "i1" });
    // grouping comparator based on the first part of the pair
    job.setOutputGroupingColumns(new String[] { "i1" });
    // the map output is LongPair, Long
    job.setMapOutputKeySchema(SchemaUtils.fromString("i1:bigint,i2:bigint"));
    Job. Fig (schemeutils. fromstring ("i2x: bigint "));
    InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    JobClient.runJob(job);
    System.exit(0);
}

```

4.3.6. Resource samples

Prerequisites

1. Prepare a Jar package of the test program. Assume the package is named `mapreduce-examples.jar`, and the local storage path is `data/resources`.
2. Prepare the test table and the resource.
 - o Create the tables:

```
create table mr_upload_src(key bigint, value string);
```

- Add the resource:

```
add jar data\resources\mapreduce-examples.jar -f;
add file data\resources\import.txt -f;
```

- The contents of import.txt:

```
1000,odps
```

Procedure

Run Upload on the odpscmd:

```
jar -resources mapreduce-examples.jar,import.txt -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Upload import.txt mr_upload_src;
```

Expected output

The content in the output table “mr_upload_src” is as follows:

```
+-----+-----+
| key | value |
+-----+-----+
| 1000 | odps |
+-----+-----+
```

Sample code

```
package com.aliyun.odps.mapred.open.example;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;

/**
 * Upload
```



```

*
* Import data from text file into table
*
**/
public class Upload {
    public static class UploadMapper extends MapperBase {
        @Override
        public void setup(TaskContext context) throws IOException{
            Record record = context.createOutputRecord();
            StringBuilder importdata = new StringBuilder();
            BufferedInputStream bufferedInput = null;
            try {
                byte[] buffer = new byte[1024];
                int bytesRead = 0;
                String filename = context.getJobConf().get("import.filename");
                bufferedInput = context.readResourceFileAsStream(filename);
                while ((bytesRead = bufferedInput.read(buffer)) != -1) {
                    String chunk = new String(buffer, 0, bytesRead);
                    importdata.append(chunk);
                }
                String lines[] = importdata.toString().split("\n");
                for (int i = 0; i < lines.length; i++) {
                    String[] ss = lines[i].split(",");
                    record.set(0, Long.parseLong(ss[0].trim()));
                    record.set(1, ss[1].trim());
                    context.write(record);
                }
            } catch (FileNotFoundException ex) {
                throw new IOException(ex);
            } catch (IOException ex) {
                throw new IOException(ex);
            } finally {
            }
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            Throws IOException {
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {

```

```
        System.err.println("Usage: Upload <import_txt> <out_table>");
        System.exit(2);
    }
    JobConf job = new JobConf();
    job.setMapperClass(UploadMapper.class);
    // Set the Resource Name, which can be obtained from jobconf in the map
    job.set("import.filename", args[0]);
    // Maponly job needs to explicitly set reducer number to 0
    job.setNumReduceTasks(0);
    job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint"));
    job.setMapOutputValueSchema(SchemaUtils.fromString("value:string"));
    InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), job);
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    Jobclient.runjob(job);
}
```

A user can set up JobConf through the following methods:

- Use JobConf interface in SDK. This method is used in the preceding example. Moreover, this is the most recommended method and is given the highest priority.
- In jar command lines, specify new JobConf file through the parameter `-conf`. This method is of the lowest priority.

4.3.7. Counter samples

Prerequisites

1. Prepare the Jar package of the test program. Assume the package is named `mapreduce-examples.jar`, and the local storage path is `data/resources`.
2. Prepare the UserDefinedCounters test table and resource.

- Create tables:

```
create table wc_in (key string, value string);
create table wc_out(key string, cnt bigint);
```

- Add resources:

```
add jar data/resources/mapreduce-examples.jar -f;
```

3. Use the tunnel command to import the data:

```
tunnel upload data wc_in;
```

The data imported into the wc_in the table wc_in, is as follows:

```
hello,odps
```

Procedure

Execute UserDefinedCounters on the odpscmd:

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.UserDefinedCounters wc_in wc_out
```

Expected output

The output of Counters is as follows:

```
Counters: 3
com.aliyun.odps.mapred.open.example.UserDefinedCounters$MyCounter
MAP_TASKS=1
REDUCE_TASKS=1
TOTAL_TASKS=2
```

The content of output table "wc_out" is as follows:

```
+-----+-----+
| key | cnt |
+-----+-----+
| hello | 1 |
| odps | 1 |
+-----+-----+
```

Sample code

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
import com.aliyun.odps.counter.Counter;
import com.aliyun.odps.counter.Counters;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.SchemaUtils;
```

```
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.data.TableInfo;
/**
 *
 * User Defined Counters
 *
 **/
public class UserDefinedCounters {
    enum MyCounter {
        TOTAL_TASKS, MAP_TASKS, REDUCE_TASKS
    }
    public static class TokenizerMapper extends MapperBase {
        private Record word;
        private Record one;
        @Override
        public void setup(TaskContext context) throws IOException{
            super.setup(context);
            Counter map_tasks = context.getCounter(MyCounter.MAP_TASKS);
            Counter total_tasks = context.getCounter(MyCounter.TOTAL_TASKS);
            map_tasks.increment(1);
            total_tasks.increment(1);
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.set(new Object[] { 1L });
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            Throws IOException {
            for (int i = 0; i < record.getColumnCount(); i++) {
                word.set(new Object[] { record.get(i).toString() });
                context.write(word, one);
            }
        }
    }
    public static class SumReducer extends ReducerBase {
        private Record result = null;
        @Override
        public void setup(TaskContext context) throws IOException{
            result = context.createOutputRecord();
            Counter reduce_tasks = context.getCounter(MyCounter.REDUCE_TASKS);
```

```

        Counter maid = context.getcounter (mycounter );
        reduce_tasks.increment(1);
        total_tasks.increment(1);
    }
    @Override
    public void reduce(Record key,Iterator<Record>values,TaskContext context)
        Throws IOException {
        Long Count = 0;
        while(values.hasNext()) {
            Record val = values.next();
            count += (Long) val.get(0);
        }
        result.set(0, key.get(0));
        result.set(1, count);
        context.write(result);
    }
}

public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err
            .println("Usage: TestUserDefinedCounters <in_table> <out_table>");
        System.exit(2);
    }
    JobConf job = new JobConf();
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
    job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
    InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    RunningJob rjob = JobClient.runJob(job);
    // After the job has completed successfully, you can get the value of the custom counter inside th
    e job
    Counters counters = rjob.getCounters();
    long m = counters.findCounter(MyCounter.MAP_TASKS).getValue();
    long r = counters.findCounter(MyCounter.REDUCE_TASKS).getValue();
    long total = counters.findCounter(MyCounter.TOTAL_TASKS).getValue();
    System.exit(0);
}
}

```

4.3.8. Grep samples

Prerequisites

1. Prepare the jar package of the test program. Assume the package is named `mapreduce-examples.jar`, and the local storage path is `data/resources`.
2. Prepare tables and resources for testing the Grep operation.
 - Create tables:

```
create table mr_src(key string, value string);
create table mr_grep_tmp (key string, cnt bigint);
create table mr_grep_out (key bigint, value string);
```

- Add resources:

```
add jar data/resources/mapreduce-examples.jar -f;
```

3. Use the tunnel command to import the data:

```
tunnel upload data mr_src;
```

The contents of data file imported into the table “mr_src” :

```
hello,odps
hello,world
```

Procedure

Execute Grep on the odpscmd:

```
jar -resources mapreduce-examples.jar -classpath data/resources/mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Grep mr_src mr_grep_tmp mr_grep_out hello;
```

Expected output

The content of output table “mr_grep_out” is as follows:

```
+-----+-----+
| key | value |
+-----+-----+
| 2 | hello |
+-----+-----+
```

Sample code

```
com.aliyun.odps.mapred.open.example.Grep mr_src mr_grep_tmp mr_grep_out hello;
```

```
package com.aliyun.odps.mapred.open.example;

import java.io.IOException;
import java.util.Iterator;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.Mapper;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.RunningJob;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;

/**
 *
 * Extracts matching regexs from input files and counts them.
 *
 */
public class Grep {
    /**
     * RegexMapper
     */
    public class RegexMapper extends MapperBase {
        private Pattern pattern;
        private int group;
        private Record word;
        private Record one;

        @Override
        public void setup(TaskContext context) throws IOException{
            JobConf job = (JobConf) context.getJobConf();
            pattern = Pattern.compile(job.get("mapred.mapper.regex"));
            group = job.getInt("mapred.mapper.regex.group", 0);
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.set(new Object[] { 1L });
        }

        @Override
```

```

public void map(long recordNum, Record record, TaskContext context) throws IOException {
    for (int i = 0; i < record.getColumnCount(); ++i) {
        String text = record.get(i).toString();
        Matcher = pattern.matcher(text);
        while (matcher.find()) {
            word.set(new Object[] { matcher.group(group) });
            context.write(word, one);
        }
    }
}

/**
 * LongSumReducer
 */
public class LongSumReducer extends ReducerBase {
    private Record result = null;

    @Override
    public void setup(TaskContext context) throws IOException {
        result = context.createOutputRecord();
    }

    @Override
    public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOException
    {
        Long count = 0;
        while(values.hasNext()) {
            Record val = values.next();
            count += (Long) val.get(0);
        }
        result.set(0, key.get(0));
        result.set(1, count);
        context.write(result);
    }
}

/**
 * A {@link Mapper} that swaps keys and values.
 */
public class InverseMapper extends MapperBase {
    private Record word;
    private Record count;

    @Override
    public void setup(TaskContext context) throws IOException{

```



```

        word = context.createMapOutputValueRecord();
        count = context.createMapOutputKeyRecord();
    }
    /**
     * The inverse function. Input keys and values are swapped.
     */
    @Override
    public void map(long recordNum, Record record, TaskContext context) throws IOException {
        word.set(new Object[] { record.get(0).toString() });
        count.set(new Object[] { (Long) record.get(1) });
        context.write(count, word);
    }
}
/**
 * IdentityReducer
 */
public class IdentityReducer extends ReducerBase {
    private Record result = null;
    @Override
    public void setup(TaskContext context) throws IOException {
        result = context.createOutputRecord();
    }
    /** Writes all keys and values directly to output. */
    @Override
    public void reduce(Record key, Iterator<Record> values, TaskContext context) throws IOException
    {
        result.set(0, key.get(0));
        while(values.hasNext()) {
            Record val = values.next();
            result.set(1, val.get(0));
            context.write(result);
        }
    }
}
public static void main(String[] args) throws Exception {
    if (args.length < 4) {
        System.err.println("Grep <inDir> <tmpDir> <outDir> <regex> [<group>]");
        System.exit(2);
    }
    JobConf grepJob = new JobConf();

```

```

grepJob.setMapperClass(RegexMapper.class);
grepJob.setReducerClass(LongSumReducer.class);
grepJob.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
grepJob.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), grepJob);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), grepJob);
// Set the regular expression for grepjob's grep
grepJob.set("mapred.mapper.regex", args[3]);
if (args.length == 5) {
    grepJob.set("mapred.mapper.regex.group", args[4]);
}
@SuppressWarnings("unused")
RunningJob rjGrep = JobClient.runJob(grepJob);
// Grepjob output as input to sortjob
JobConf sortJob = new JobConf();
sortJob.setMapperClass(InverseMapper.class);
sortJob.setReducerClass(IdentityReducer.class);
sortJob.setMapOutputKeySchema(SchemaUtils.fromString("count:bigint"));
sortJob.setMapOutputValueSchema(SchemaUtils.fromString("word:string"));
InputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), sortJob);
OutputUtils.addTable(TableInfo.builder().tableName(args[2]).build(), sortJob);
sortJob.setNumReduceTasks(1); // write a single file
sortJob.setOutputKeySortColumns(new String[] { "count" });
@SuppressWarnings("unused")
RunningJob rjSort = JobClient.runJob(sortJob);
}
}

```

4.3.9. Join samples

The MaxCompute MapReduce framework does not support join logic on its own. Therefore, you have to apply join samples of the data in your own map/reduce function which requires you to do some extra work.

Suppose, to join two tables (Key bigint, value string) and (key bigint, value string), the output table is chain bigint (value1 string, value2 string), where value1 and value2 are the values of the scanner.

Prerequisites

1. Prepare the jar package for the test program, assuming the name is maid and the local storage path is data \ resources.
2. Prepare tables and resources for testing the Join operation.

- Create tables:

```
create table mr_Join_src1(key bigint, value string);
create table mr_Join_src2(key bigint, value string);
create table mr_Join_out(key bigint, value1 string,value2 string);
```

- Add resources:

```
add jar data\resources\mapreduce-examples.jar -f;
```

3. Run tunnel to import the data:

```
tunnel upload data1 mr_Join_src1;
tunnel upload data2 mr_Join_src2;
```

Import the contents of the maid data as follows:

```
1, hello
2, ODPS
```

Import the contents of the maid data as follows:

```
1, ODPS
3,hello
4, ODPS
```

Procedure

Join in odpscmd as follows:-

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Join mr_Join_src1 mr_Join_src2 mr_Join_out;
```

Expected output

After the job is completed successfully, the contents of the table maid are output, as follows:

```
+-----+-----+-----+
| key | value1 | value2 |
+-----+-----+-----+
| 1 | hello | odps |
+-----+-----+-----+
```

Sample code

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.ArrayList;
```

```
import java.util.arraylist;
import java.util.Iterator;
import java.util.List;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.aliyun.ODPS.Data.record;-
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
 * Join, mr_Join_src1/mr_Join_src2(key bigint, value string), mr_Join_out(key
 * bigint, value1 string, value2 string)
 *
 */
public class Join {
    public static final Log LOG = LogFactory.getLog(Join.class);
    public static class JoinMapper extends MapperBase {
        private Record mapkey;
        private Record mapvalue;
        private long tag;
        @Override
        public void setup(TaskContext context) throws IOException{
            mapkey = context.createMapOutputKeyRecord();
            mapvalue = context.createMapOutputValueRecord();
            tag = context.getInputTableInfo().getLabel().equals("left") ? 0: 1;
        }
        @Override
        public void map(long key,Record record, TaskContext context)
            Throws ioexception {
            mapkey.set(0,record.get(0));
            mapkey.set(1,tag);
            for (int i = 1; i< record.getColumnCount();i++) {
                mapvalue.set(i -1, record.get(i));
            }
            context.write(mapkey,mapvalue);
        }
    }
}
```

```

    }

    public static class JoinReducer extends ReducerBase {
        private Record result = null;

        @Override
        public void setup(TaskContext context) throws IOException{
            result = context.createOutputRecord();
        }

        //Reduce function all records for each input will be the same key
        @Override
        public void reduce(Record key,Iterator<Record>values,TaskContext context)
            Throws ioexception {
            long k = key.getBigint(0);
            List<Object[]> leftValues = new ArrayList<Object[]>();

            //Is a key + tag combination because it is set up, this ensures that record data in the left table is
            in front of the input record for the reduce function.

            while(values.hasNext()) {
                Record value = values.next();
                long tag = (Long)key.get(1);
                //The data for the left table is first cached into memory
                if (tag == 0) {
                    leftValues.add(value.toArray().clone());
                }else {
                    //The data that touches the right table is output by a join with all the data on the left table, th
                    e data for the left table is all in memory.
                }
            }
            //This implementation is just a functional display with relatively low performance and is not recommen
            ded for practical production.

            for (Object[] leftValue :leftValues) {
                int index = 0;
                result.set(index++,k);
                for (int i = 0;i<leftValue.length;i++) {
                    result.set(index++,leftValue[i]);
                }
                for (int i = 0;i< value.getColumnCount();i++) {
                    result.set(index++,value.get(i));
                }
                context.write(result);
            }
        }
    }
}

```

```
public static void main(String[] args) throws Exception {
    if (args.length != 3) {
        System.err.println("Usage: Join <input table1> <input table2> <out>");
        System.exit(2);
    }
    JobConf job = new JobConf();
    job.setMapperClass(JoinMapper.class);
    job.setReducerClass(JoinReducer.class);
    job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,tag:bigint"));
    job.setMapOutputValueSchema(SchemaUtils.fromString("value:string"));
    job.setPartitionColumns(new String[]{"key"});
    job.setOutputKeySortColumns(new String[]{"key", "tag"});
    job.setOutputGroupingColumns(new String[]{"key"});
    job.setNumReduceTasks(1);
    InputUtils.addTable(TableInfo.builder().tableName(args[0]).label("left").build(), job);
    InputUtils.addTable(TableInfo.builder().tableName(args[1]).label("right").build(), job);
    OutputUtils.addTable(TableInfo.builder().tableName(args[2]).build(), job);
    Jobclient.runjob(job);
}
```

4.3.10. Sleep samples

Prerequisites

1. Prepare the jar package of the test program. Assume the package is named `mapreduce-examples.jar`, and the local storage path is `data/resources`.
2. Prepare resources for testing the SleepJob operation.

```
Add jar data \ resources \ mapreduce-examples.jar-f;
```

Procedure

Run Sleep on the odpscmd is as follows:

```
jar -resources mapreduce-examples.jar -classpath data/resources/mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Sleep 10;
jar -resources mapreduce-examples.jar -classpath data/resources/mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Sleep 100;
```

Expected output

The job runs successfully. The run time of different sleep durations can be compared to determine the effect.

Sample code

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.mapperbase;
import com.aliyun.odps.mapred.conf.JobConf;
public class Sleep {
    private static final String SLEEP_SECS = "sleep.secs";
    public static class MapperClass extends MapperBase {
        // Because the data is not entered, the map function is not executed, and the related logic can only
        be written in setup
        @Override
        public void setup(TaskContext context) throws IOException {
            try {
                // Get the number of sleep seconds set in jobconf to sleep
                Thread.sleep(context.getJobConf().getInt(SLEEP_SECS, 1) * 1000);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
    public static void main(String[] args) throws Exception {
        if (args.length != 1) {
            System.err.println("Usage: Sleep <sleep_secs>");
            System.exit(-1);
        }
        JobConf job = new JobConf();
        job.setMapperClass(MapperClass.class);
        // This instance is also a maponly, so you need to set the reducer number to 0.
        job.setNumReduceTasks(0);
        // Because there is no input table, the number of mapper needs to be specified explicitly by the use
        r
        job.setNumMapTasks(1);
        job.set(SLEEP_SECS, args[0]);
        JobClient.runJob(job);
    }
}
```

4.3.11. Unique samples

Prerequisites

1. Prepare the JAR package of the test program. Assume the package is named `mapreduce-examples.jar`, and the local storage path is `data\resources`.
2. Prepare tables and resources for testing the Unique operation.

- Create tables:

```
create table ss_in(key bigint, value bigint);
create table ss_out(key bigint, value bigint);
```

- Add resources:

```
add jar data\resources\mapreduce-examples.jar -f;
```

3. Use the tunnel command to import the data.

```
tunnel upload data ss_in;
```

The contents of data file are imported into the table `ss_in`.

```
1,1
1,1
2,2
2,2
```

Procedure

Run Unique on the `odpscmd`, as follows:

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Unique ss_in ss_out key;
```

Expected output

The content of output table `ss_out` is as follows:

```
+-----+-----+
| key | value |
+-----+-----+
| 1 | 1 |
| 2 | 2 |
+-----+-----+
```

Sample code

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
```

```

import java.util. iterator;
import com.aliyun. ODPS. Data. record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
 * Unique Remove duplicate words
 *
 **/
public class Unique {
    public static class OutputSchemaMapper extends MapperBase {
        private Record key;
        private Record value;
        @Override
        public void setup(TaskContext context) throws IOException{
            key = context.createMapOutputKeyRecord();
            value = context.createMapOutputValueRecord();
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            Throws IOException {
            long left = 0;
            long right = 0;
            if (record.getColumnCount() > 0) {
                left = (Long) record.get(0);
                if (record.getColumnCount() > 1) {
                    right = (Long) record.get(1);
                }
                key.set(new Object[] { (Long) left, (Long) right });
                value.set(new Object[] { (Long) left, (Long) right });
                context.write(key, value);
            }
        }
    }
    public static class OutputSchemaReducer extends ReducerBase {

```

```

private Record result = null;

@Override
public void setup(TaskContext context) throws IOException{
    result = context.createOutputRecord();
}

@Override
public void reduce(Record key,Iterator<Record>values,TaskContext context)
    Throws IOException {
    result.set(0, key.get(0));
    while(values.hasNext()) {
        Record value = values.next();
        result.set(1, value.get(1));
    }
    context.write(result);
}
}

public static void main(String[] args) throws Exception {
    if (args.length > 3 || args.length < 2) {
        System.err.println("Usage: unique <in> <out> [key|value|all]");
        System.exit(2);
    }
    String ops = "all";
    if (args.length == 3) {
        Ops = ARGS [2];
    }
    // Reduce input grouping is determined by the settings of the scanner, this parameter if it is not s
    et
    /Default is mapoutputkeyschema
    // Key Unique
    if (ops.equals("key")) {
        JobConf job = new JobConf();
        job.setMapperClass(OutputSchemaMapper.class);
        job.setReducerClass(OutputSchemaReducer.class);
        job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setPartitionColumns(new String[] { "key" });
        job.setOutputKeySortColumns(new String[] { "key", "value" });
        job.setOutputGroupingColumns(new String[] { "key" });
        job.set("tablename2", args[1]);
        job.setNumReduceTasks(1);
        job.set("table-connector", O);
    }
}

```

```

        job.setInt("table.counter", 0);
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        Jobclient.runjob(job);
    }

    // Key&Value Unique
    if (ops.equals("all")) {
        JobConf job = new JobConf();
        job.setMapperClass(OutputSchemaMapper.class);
        job.setReducerClass(OutputSchemaReducer.class);
        job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setPartitionColumns(new String[] { "key" });
        job.setOutputKeySortColumns(new String[] { "key", "value" });
        job.setOutputGroupingColumns(new String[] { "key", "value" });
        Job.set("tablename2", args[1]);
        job.setNumReduceTasks(1);
        job.setInt("table.counter", 0);
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        Jobclient.runjob(job);
    }

    // Value Unique
    if (ops.equals("value")) {
        JobConf job = new JobConf();
        job.setMapperClass(OutputSchemaMapper.class);
        job.setReducerClass(OutputSchemaReducer.class);
        job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("key:bigint,value:bigint"));
        job.setPartitionColumns(new String[] { "value" });
        job.setOutputKeySortColumns(new String[] { "value" });
        job.setOutputGroupingColumns(new String[] { "value" });
        job.set("tablename2", args[1]);
        job.setNumReduceTasks(1);
        job.setInt("table.counter", 0);
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        Jobclient.runjob(job);
    }
}
}
}

```

4.3.12. Sort samples

Prerequisites

1. Prepare the Jar package of the test program. Assume the package is named `mapreduce-examples.jar`, and the local storage path is `data\resources`.
2. Prepare tables and resources for testing the SORT operation.
 - Create tables:

```
create table ss_in(key bigint, value bigint);
create table ss_out(key bigint, value bigint);
```

- Add resources:

```
add jar data\resources\mapreduce-examples.jar -f;
```

3. Use the tunnel command to import the data.

```
tunnel upload data ss_in;
```

The contents of data file in the table `ss_in` are as follows:

```
2,1
1,1
3,1
```

Procedure

Run Sort on the `odpscmd`, as follows:

```
jar -resources mapreduce-examples.jar -classpath data\resources\mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.Sort ss_in ss_out;
```

Expected output

The content of the output table `ss_out` is as follows:

```
+-----+-----+
| key | value |
+-----+-----+
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
+-----+-----+
```

Sample code

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Date;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.TaskContext;
import com.aliyun.odps.mapred.conf.JobConf;
import com.aliyun.odps.mapred.example.lib.IdentityReducer;
import com.aliyun.odps.mapred.utils.InputUtils;
import com.aliyun.odps.mapred.utils.OutputUtils;
import com.aliyun.odps.mapred.utils.SchemaUtils;
/**
 * This is the trivial map/reduce program that does absolutely nothing other
 * than use the framework to fragment and sort the input values.
 */
public class Sort {
    static int printUsage() {
        System.out.println("sort <input> <output>");
        return -1;
    }
    /**
     * Implements the identity function, mapping record's first two columns to
     * outputs.
     */
    public static class IdentityMapper extends MapperBase {
        private Record key;
        private Record value;
        @Override
        public void setup(TaskContext context) throws IOException{
            key = context.createMapOutputKeyRecord();
            value = context.createMapOutputValueRecord();
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            Throws IOException {
            Key.set (new object [] {(long) record.get (0 )});
```

```

        value.set(new Object[] { (Long) record.get(1) });
        context.write(key, value);
    }
}
/**
 * The main driver for sort program. Invoke this method to submit the
 * map/reduce job.
 *
 * @throws IOException
 * When there is communication problems with the job tracker.
 */
public static void main(String[] args) throws Exception {
    JobConf jobConf = new JobConf();
    jobConf.setMapperClass(IdentityMapper.class);
    jobConf.setReducerClass(IdentityReducer.class);
    // For global order, the number of reducers is set to 1, all the data will be concentrated on a reducer.
    // Can be used only for small volumes of data, which need to be considered in other ways, such as terasort.
    jobConf.setNumReduceTasks(1);
    jobConf.setMapOutputKeySchema(SchemaUtils.fromString("key: bigint"));
    jobConf.setMapOutputValueSchema(SchemaUtils.fromString("value: bigint"));
    InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), jobConf);
    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), jobConf);
    Date starttime = new Date();
    System.out.println("Job started: " + starttime);
    JobClient.runJob(jobConf);
    Date end_time = new Date();
    System.out.println("Job ended: " + end_time);
    System.out.println("The job took "
        + (end_time.getTime() - starttime.getTime()) / 1000 + " seconds.");
}
}

```

4.3.13. Partition samples

The following example takes Partition as input and output.

Example 1:

```
public static void main(String[] args) throws Exception {
    JobConf job = new JobConf();

    LinkedHashMap<String, String> input = new LinkedHashMap<String, String>();
    input.put("pt", "123456");
    InputUtils.addTable(TableInfo.builder().tableName("input_table").partSpec(input).build(), job);
    LinkedHashMap<String, String> output = new LinkedHashMap<String, String>();
    output.put("ds", "654321");
    OutputUtils.addtable (tableinfo. builder (). tablename ("output_table "). partspec (output ). build (),
    job );
    JobClient.runJob(job);
}
```

Example 2:


```

package com.aliyun.odps.mapred.open.example;

public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("Usage: WordCount <in_table> <out_table>");
        System.exit(2);
    }

    JobConf job = new JobConf();
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(SumCombiner.class);
    job.setReducerClass(SumReducer.class);
    job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
    job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
    Account account = new AliyunAccount("my_access_id", "my_access_key");
    Odps odps = new Odps(account);
    odps.setEndpoint("odps_endpoint_url");
    odps.setDefaultProject("my_project");
    Table table = odps.tables().get(tblname);
    TableInfoBuilder builder = TableInfo.builder().tableName(tblname);
    for (Partition p : table.getPartitions()) {
        if (applicable(p)) {
            LinkedHashMap<String, String> partSpec = new LinkedHashMap<String, String>();
            for (String key : p.getPartitionSpec().keys()) {
                partSpec.put(key, p.getPartitionSpec().get(key));
            }

            InputUtils.addTable(builder.partSpec(partSpec).build(), conf);
        }
    }

    OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
    Jobclient.runjob(job);
}

```

Note

- The preceding example combines the MaxCompute SDK and MapReduce SDK to achieve a MapReduce task.
- The code cannot be compiled and is only an example of main functions.
- The Applicable function is user logic that determines whether the Partition can be used as the input of MapReduce job.

4.3.14. Pipeline samples

Prerequisites

1. Prepare the jar package of the test program. Assume the package is named `mapreduce-examples.jar`, and the local storage path is `data/resources`.
2. Prepare tables and resources for testing the the WordCountPipeline operation.

- Create tables:

```
create table wc_in (key string, value string);
create table wc_out(key string, cnt bigint);
```

- Add resources:

```
add jar data/resources/mapreduce-examples.jar -f;
```

3. Use the tunnel command to import the data:

```
tunnel upload data wc_in;
```

The data imported into the `wc_in` in the table `wc_in` is as follows:

```
hello,odps
```

Procedure

Run WordCountPipeline on the `odpscmd`, as follows:

```
jar -resources mapreduce-examples.jar -classpath data/resources/mapreduce-examples.jar
com.aliyun.odps.mapred.open.example.WordCountPipeline wc_in wc_out;
```

Expected output

The content of output table `wc_out` is as follows:

```
+-----+-----+
| key | cnt |
+-----+-----+
| hello | 1 |
| odps | 1 |
+-----+-----+
```

Sample code

```
package com.aliyun.odps.mapred.open.example;
import java.io.IOException;
import java.util.Iterator;
```

```
import com.aliyun.odps.Column;
import com.aliyun.odps.OdpsException;
import com.aliyun.odps.OdpsType;
import com.aliyun.ODPS.Data.record;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.mapred.Job;
import com.aliyun.odps.mapred.MapperBase;
import com.aliyun.odps.mapred.ReducerBase;
import com.aliyun.odps.pipeline.Pipeline;
public class WordCountPipelineTest {
    public static class TokenizerMapper extends MapperBase {
        Record word;
        Record one;
        @Override
        public void setup(TaskContext context) throws IOException{
            word = context.createMapOutputKeyRecord();
            one = context.createMapOutputValueRecord();
            one.setBigint(0, 1L);
        }
        @Override
        public void map(long recordNum, Record record, TaskContext context)
            Throws IOException {
            for (int i = 0; i < record.getColumnCount(); i++) {
                String[] words = record.get(i).toString().split("\\s+");
                for (String w : words) {
                    word.setString(0, w);
                    context.write(word, one);
                }
            }
        }
    }
    public static class SumReducer extends ReducerBase {
        private Record value;
        @Override
        public void setup(TaskContext context) throws IOException{
            value = context.createOutputValueRecord();
        }
        @Override
        public void reduce(Record key,Iterator<Record>values,TaskContext context)
            Throws IOException {
            Long Count = 0;
```

```

        while(values.hasNext()) {
            Record val = values.next();
            count += (Long) val.get(0);
        }
        value.set(0, count);
        context.write(key, value);
    }
}

public static class IdentityReducer extends ReducerBase {
    private Record result;

    @Override
    public void setup(TaskContext context) throws IOException{
        result = context.createOutputRecord();
    }

    @Override
    public void reduce(Record key,Iterator<Record>values,TaskContext context)
        Throws IOException {
        while (values.hasNext()) {
            result.set(0, key.get(0));
            result.set(1, values.next().get(0));
            context.write(result);
        }
    }
}

public static void main(String[] args) throws OdpsException {
    if (args.length != 2) {
        System.err.println("Usage: WordCountPipeline <in_table> <out_table>");
        System.exit(2);
    }
    Job job = new Job();
    /**
     * In the process of constructing pipeline, if you do not specify mapper's OutputKeySortColumns,
     * PartitionColumns, OutputGroupingColumns,
     * the framework defaults to its OutputKey as the default configuration for the three
     */
    Pipeline pipeline = Pipeline.builder()
        .Addmapper (maid. Class)
        .setOutputKeySchema(
            new Column[] { new Column("word", OdpsType.STRING) })
        .setOutputValueSchema(
            new Column[] { new Column("count", OdpsType.BIGINT) })

```

```

        new Column[] { new Column("count", OdpsType.BIGINT) })
        .setOutputKeySortColumns(new String[] { "word" })
        .setPartitionColumns(new String[] { "word" })
        .setOutputGroupingColumns(new String[] { "word" })
        .addReducer(SumReducer.class)
        .setOutputKeySchema(
            new Column[] { new Column("word", OdpsType.STRING) })
        .setOutputValueSchema(
            new Column[] { new Column("count", OdpsType.BIGINT) })
        .addReducer(IdentityReducer.class).createPipeline();
// Set pipeline to jobconf and jobconf if you need to set the assembler
job.setPipeline(pipeline);
// Set table information for Input Output
job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addOutput(TableInfo.builder().tableName(args[1]).build());
// Job submit and wait for end
job.submit();
job.waitForCompletion();
System.exit(job.isSuccessful() == true ? 0 : 1);
}
}

```

4.4. Java SDK

4.4.1. Java SDK

This article introduces common MapReduce interfaces.

If you are using Maven, you can search "odps-sdk-mapred" from [Maven Library](#) to get the required Java SDK (available in different versions). The configuration is as follows:

```

<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-mapred</artifactId>
  <version>0.20.7-public</version>
</dependency>

```

Interface	Description

Interface	Description
MapperBase	The user-defined Map function is required to inherit from this class. It processes the record object of the input table, processes the object into key value and outputs the value to the Reduce stage, or outputs result record to the result table without passing through the Reduce stage. Jobs that do not pass through the Reduce stage, but directly outputs computation results are called Map-Only job.
ReducerBase	Your customized Reduce function must inherit from this class. The set of Values associated with a Key is reduced.
TaskContext	It is one of the input parameters of multiple member functions in MapperBase and ReducerBase. Contains contextual information about tasks.
JobClient	It is used for submitting and managing jobs. The submission mode includes blocking (synchronous) mode or non-blocking (asynchronous) mode.
RunningJob	Indicates object in job running and used for tracing MapReduce job instance during the job running process.
JobConf	Describes configuration of a MapReduce task. The JobConf object is generally defined in the main program (main function), then jobs are submitted by JobClient to MaxCompute.

MapperBase

Main function interfaces are as follows.

Interface	Description
void cleanup(TaskContext context)	The Map method is called after the map stage ends.
void map(long key, Record record, TaskContext context)	The Map method processes records of the input table.
void setup(TaskContext context)	The Map method is called before the map stage begins.

ReducerBase

Main function interfaces are as follows.

Interface	Description
void cleanup(TaskContext context)	The Reduce method is called after the reduce stage ends.
void reduce(Record key, Iterator<Record > values, TaskContext context)	The Reduce method processes input table records.
void setup(TaskContext context)	The Reduce method is called before the reduce stage begins.

TaskContext

Main function interfaces are as follows.

Interface	Description
TableInfo[] getOutputTableInfo()	Gets output table information.
Record createOutputRecord()	Creates the record object of the default output table.
Record createOutputRecord(String label)	Creates the record object of the output table with a specified label.
Record createMapOutputKeyRecord()	Creates the record object of Key output by Map.
Record createMapOutputValueRecord()	Creates the record object of Value output by Map.
void write(Record record)	Writes record to default output and is used for writing output data by Reduce client, and can be called on the Reduce client multiple times.
void write(Record record, String label)	Writes record to the given label output and is used for writing output data by Reduce client, and can be called on the Reduce client multiple times.
void write(Record key, Record value)	Map writes record for an intermediate result. It can be called in Map function and called on the Map client multiple times.
BufferedInputStream readResourceFileAsStream(String resourceName)	Reads file type resource.
Iterator<Record > readResourceTable(String resourceName)	Reads table type resource.
Counter getCounter(Enum<? > > name)	Gets the Counter object with the specified name.
Counter getCounter(String group, String name)	Gets the Counter object with specified name and the group name.
void progress()	Reports heartbeat information to the MapReduce framework. If a user's method takes a long time to process, and no framework is called in the process, this method can be called to avoid task timeout. Timeout of the framework is 600s by default.

 Notice

- MaxCompute TaskContext interface provides the progress function, however, this function is to prevent the Worker from being terminated as it runs for long time and the framework considers it as a timeout Worker. This interface is similar to sending heartbeat information to the framework, but does not report the progress of the Worker.
- The default timeout schedule of MaxCompute MapReduce Worker is 10 minutes (system default, cannot be controlled by the user). If the schedule exceeds 10 minutes and Worker is unable to send heartbeat information to the framework (not to call progress interface), the framework is forced to stop this Worker and MapReduce task fails and exits. We recommend calling the progress interface regularly in Mapper/Reducer functions to prevent the worker from being terminated by the framework.

JobConf

Main function interfaces are as follows:

Interface	Description
<code>void setResources(String resourceNames)</code>	Declares resources used in this job. Only the declared resource can be read by TaskContext object during Mapper/Reducer running process.
<code>void setMapOutputKeySchema(Column[] schema)</code>	Sets the Key attribute output from Mapper to Reducer.
<code>void setMapOutputValueSchema(Column[] schema)</code>	Sets the Value attribute output from Mapper to Reducer.
<code>void setOutputKeySortColumns(String[] cols)</code>	Sets key sort columns output from Mapper to Reducer.
<code>void setOutputGroupingColumns(String[] cols)</code>	Sets Key grouping columns.
<code>void setMapperClass(Class<? extends Mapper > theClass)</code>	Sets Mapper function of the job.
<code>void setPartitionColumns(String[] cols)</code>	Sets the partition column specified in the job. The default is all columns of Key output by Mapper.
<code>void setReducerClass(Class<? extends Reducer theClass)</code>	Sets Reducer of the job.
<code>void setCombinerClass(Class<? extends Reducer theClass)</code>	Sets combiner of the job, running on Map client. Its function is similar to performing Reduce operation on the identical local Key values by a single Map.
<code>void setSplitSize(long size)</code>	Sets the size of input slice. Unit: MB. The default value is 640.

Interface	Description
<code>void setNumReduceTasks(int n)</code>	Sets the number of Reducer tasks. The default is 1/4 of Mapper tasks.
<code>void setMemoryForMapTask(int mem)</code>	Sets the memory size of single Worker in the Mapper task. Unit: MB. The default value is 2048.
<code>void setMemoryForReduceTask(int mem)</code>	Sets the memory size of single Worker for Reducer task. Unit: MB. The default value is 2048.

 Note

- Usually, GroupingColumns are included in KeySortColumns, while KeySortColumns and PartitionColumns are included in the Key.
- In the Map side, mappers' output records are distributed to reducers according to the hash values computed using PartitionColumns, and then sorted by KeySortColumns.
- In the Reduce side, after being sorted by KeySortColumns, input records are grouped as input groups of the reduce function sequentially. In other words, records with the same GroupingColumns values are treated as the same input group.

JobClient

Main function interfaces are as follows:

Interface	Description
<code>static RunningJob runJob(JobConf job)</code>	Returns immediately after submitting a MapReduce job in a synchronous (blocking) mode.
<code>static RunningJob submitJob(JobConf job)</code>	Returns immediately after submitting a MapReduce job in an asynchronous (non-blocking) mode.

RunningJob

Main function interfaces are as follows.

Interface	Description
<code>String getInstanceID()</code>	Gets an instance ID for checking run log and job management.
<code>boolean isComplete()</code>	Checks whether job is complete.
<code>boolean isSuccessful()</code>	Checks whether job instance is successful.
<code>void waitForCompletion()</code>	Waits until job instance is complete. It is typically used for jobs submitted in asynchronous mode.

Interface	Description
JobStatus getJobStatus()	Checks job instance status.
void killJob()	Ends the job.
Counters getCounters()	Gets Counter information.

InputUtils

Main function interfaces are as follows:

Interface	Description
static void addTable(TableInfo table, JobConf conf)	Adds table to the task input. It can be called multiple times. The new added table is added to input queue in an append mode.
static void setTables(TableInfo [] tables, JobConf conf)	Adds tables to the task input.

OutputUtils

Main function interfaces are as follows:

Interface	Description
static void addTable(TableInfo table, JobConf conf)	Adds table to the task output. It can be called multiple times. Also, adds the new added table to output queue in an append mode.
static void setTables(TableInfo [] tables, JobConf conf)	Adds multiple tables to the task output.

Pipeline

Pipeline is the subject of **MR2**. It can be constructed by Pipeline.builder. Pipelines are as follows:

```

public Builder addMapper(Class<? extends Mapper> mapper)
public Builder addMapper(Class<? extends Mapper> mapper,
    column [] keyschema, column [] valueschema, string [] sortcols,
    SortOrder [] order, string [] partcols,
    Class<? extends Partitioner> theClass, String[] groupCols)
public Builder addReducer(Class<? extends Reducer> reducer)
public Builder addReducer(Class<? extends Reducer> reducer,
    column [] keyschema, column [] valueschema, string [] sortcols,
    SortOrder [] order, string [] partcols,
    Class<? extends Partitioner> theClass, String[] groupCols)
public setoutputkeyschema builder (Column [] keyschema)
public setoutputvalueschema builder (Column [] valueschema)
public setoutputkeysortcolumns builder (String [] sortcols)
public setoutputkeysortorder builder (Sortorder [] order)
public setpartitioncolumns builder (String [] partcols)
public Builder setPartitionerClass(Class<? extends Partitioner> theClass)
void setOutputGroupingColumns(String[] cols)

```

Example:

```

job job = new job ();
pipeline pipeline = pipeline. builder ()
    . addmapper (TokenizerMapper. class)
    . setoutputkeyschema (
        new column [] {new column ("word", OdpsType. string)})
    . setoutputvalueschema (
        new column [] {new column ("count", OdpsType. bigint)})
    . addreducer (SumReducer. class)
    . setoutputkeyschema (
        new column [] {new column ("count", OdpsType. bigint)})
    . setoutputvalueschema (
        new column [] {new column ("word", OdpsType. string),
            new column ("count", OdpsType. bigint)})
    . addreducer (IdentityReducer. class). createPipeline ();
job. setpipeline (pipeline);
job. addinput (...);
job. addoutput (...);
job. submit ();

```

As shown in the preceding example, a user can construct a Map in the main class, and then consecutively get MapReduce tasks of two Reduces. If you are familiar with the basic functions of MapReduce, then you can use MR2 as well, as the functions are similar.

 Note

- Specifically, we recommend that users must complete the configuration of MapReduce task by JobConf,
- as JobConf can get MapReduce task of single Reduce only after configuring Map.

Data Type

The data types supported in MapReduce include: BIGINT, STRING, DOUBLE, BOOLEAN, and DATETIME. MaxCompute between MaxCompute data types and Java types are as follows:

MaxCompute SQL Type	Java Type
Bigint	Long
String	String
Double	Double
Boolean	Boolean
Datetime	Date
Decimal	BigDecimal

4.4.2. Overview of compatible versions of the SDK

A detailed list of maxcompute compatible versions of mapreduce compatibility with hadoop mapreduce, as shown in the following table:

Type	Interface	Is it compatible?
Mapper	void map(KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Mapper.Context context)	Yes
Mapper	void run(org.apache.hadoop.mapreduce.Mapper.Context context)	Yes
Mapper	void setup(org.apache.hadoop.mapreduce.Mapper.Context context)	Yes
Reducer	Void cleanup (Org. Apache. hadoop. mapreduce. reducer. Context Context)	Yes

Type	Interface	Is it compatible?
Reducer	void reduce(KEYIN key, VALUEIN value, org.apache.hadoop.mapreduce.Reducer.Context context)	Yes
Reducer	void run(org.apache.hadoop.mapreduce.Reducer.Context context)	Yes
Reducer	void setup(org.apache.hadoop.mapreduce.Reducer.Context context)	Yes
Partitioner	int getPartition(KEY key, VALUE value, int numPartitions)	Yes
Mapcontext (inheritance)	InputSplit getInputSplit()	No, throw exception
ReduceContext	nextKey()	Yes
ReduceContext	getValues()	Yes
TaskInputOutputContext	getCurrentKey()	Yes
TaskInputOutputContext	getCurrentValue()	Yes
TaskInputOutputContext	getOutputCommitter()	No, throw exception
TaskInputOutputContext	nextKeyValue()	Yes
TaskInputOutputContext	write(KEYOUT key, VALUEOUT value)	Yes
TaskAttemptContext	getCounter(Enum < > counterName)	Yes
TaskAttemptContext	getCounter(String groupName, String counterName)	Yes
TaskAttemptContext	setStatus(String msg)	Empty implementation
TaskAttemptContext	getStatus()	Empty implementation
TaskAttemptContext	getTaskAttemptID()	No, throw exception

Type	Interface	Is it compatible?
TaskAttemptContext	getProgress()	No, throw exception
TaskAttemptContext	progress()	Yes
Job	addArchiveToClassPath(Path archive)	No
Job	addCacheArchive(Uri uri)	No
Job	addCacheFile(Uri uri)	No
Job	addFileToClassPath(Path file)	No
Job	cleanupProgress()	No
Job	createSymlink()	No, throw exception
Job	failTask(TaskAttemptID taskId)	No
Job	getCompletionPollInterval(Configuration conf)	Empty implementation
Job	getCounters()	Yes
Job	getFinishTime()	Yes
Job	getHistoryUrl()	Yes
Job	getInstance()	Yes
Job	getInstance(Cluster ignored)	Yes
Job	getInstance(Cluster ignored, Configuration conf)	Yes
Job	getInstance(Configuration conf)	Yes
Job	getInstance(Configuration conf, String jobName)	Empty implementation
Job	getInstance(JobStatus status, Configuration conf)	No, throw exception
Job	getJobFile()	No, throw exception

Type	Interface	Is it compatible?
Job	getJobName()	Empty implementation
Job	getJobState()	No, throw exception
Job	getPriority()	No, throw exception
Job	getProgressPollInterval(Configuration conf)	Empty implementation
Job	getReservationId()	No, throw exception
Job	getSchedulingInfo()	No, throw exception
Job	getStartTime()	Yes
Job	getStatus()	No, throw exception
Job	getTaskCompletionEvents(int startFrom)	No, throw exception
Job	getTaskCompletionEvents(int startFrom, int numEvents)	No, throw exception
Job	getTaskDiagnostics(TaskAttemptID taskid)	No, throw exception
Job	getTaskOutputFilter(Configuration conf)	No, throw exception
Job	getTaskReports(TaskType type)	No, throw exception
Job	getTrackingURL()	Yes
Job	isComplete()	Yes
Job	isRetired()	No, throw exception
Job	isSuccessful()	Yes

Type	Interface	Is it compatible?
Job	isUber()	Empty implementation
Job	killJob()	Yes
Job	killTask(TaskAttemptID taskId)	No
Job	mapProgress()	Yes
Job	monitorAndPrintJob()	Yes
Job	reduceProgress()	Yes
Job	setCacheArchives(URI[] archives)	No, throw exception
Job	setCacheFiles(URI[] files)	No, throw exception
Job	setCancelDelegationTokenUponJobCompletion(boolean value)	No, throw exception
Job	setCombinerClass(Class<? extends Reducer> cls)	Yes
Job	setCombinerKeyGroupingComparatorClass(Class<? extends RawComparator> cls)	Yes
Job	setGroupingComparatorClass(Class<? extends RawComparator> cls)	Yes
Job	setInputFormatClass(Class<? extends InputFormat> cls)	Empty implementation
Job	setJar(String jar)	Yes
Job	setJarByClass(Class<? > cls)	Yes
Job	setJobName(String name)	Empty implementation
Job	setJobSetupCleanupNeeded(boolean needed)	Empty implementation
Job	setMapOutputKeyClass(Class<? > theClass)	Yes

Type	Interface	Is it compatible?
Job	setMapOutputValueClass(Class<? > theClass)	Yes
Job	setMapperClass(Class<? extends Mapper> cls)	Yes
Job	setMapSpeculativeExecution(boolean speculativeExecution)	Empty implementation
Job	setMaxMapAttempts(int n)	Empty implementation
Job	setMaxReduceAttempts(int n)	Empty implementation
Job	setNumReduceTasks(int tasks)	Yes
Job	setOutputFormatClass(Class<? extends OutputFormat> cls)	No, throw exception
Job	setOutputKeyClass(Class<? > theClass)	Yes
Job	setOutputValueClass(Class<? > theClass)	Yes
Job	setPartitionerClass(Class<? extends Partitioner> cls)	Yes
Job	setPriority(JobPriority priority)	No, throw exception
Job	setProfileEnabled(boolean newValue)	Empty implementation
Job	setProfileParams(String value)	Empty implementation
Job	setProfileTaskRange(boolean isMap, String newValue)	Empty implementation
Job	setReducerClass(Class<? extends Reducer> cls)	Yes
Job	setReduceSpeculativeExecution(boolean speculativeExecution)	Empty implementation

Type	Interface	Is it compatible?
Job	setReservationId(ReservationId reservationId)	No, throw exception
Job	setSortComparatorClass(Class<? extends RawComparator> cls)	No, throw exception
Job	setSpeculativeExecution(boolean speculativeExecution)	Yes
Job	setTaskOutputFilter(Configuration conf, org.apache.hadoop.mapreduce.Job.TaskStatusFilter newValue)	No, throw exception
Job	setupProgress()	No, throw exception
Job	setUser(String user)	Empty implementation
Job	setWorkingDirectory(Path dir)	Empty implementation
Job	submit()	Yes
Job	toString()	No, throw exception
Job	waitForCompletion(boolean verbose)	Yes.
Task Execution & Environment	mapreduce.map.java.opts	Empty implementation
Task Execution & Environment	mapreduce.reduce.java.opts	Empty implementation
Task Execution & Environment	mapreduce.map.memory.mb	Empty implementation
Task Execution & Environment	mapreduce.reduce.memory.mb	Empty implementation
Task Execution & Environment	mapreduce.task.io.sort.mb	Empty implementation

Type	Interface	Is it compatible?
Task Execution & Environment	mapreduce.map.sort.spill.percent	Empty implementation
Task Execution & Environment	mapreduce.task.io.soft.factor	Empty implementation
Task Execution & Environment	mapreduce.reduce.merge.inmem.thresholds	Empty implementation
Task Execution & Environment	mapreduce.reduce.shuffle.merge.percent	Empty implementation
Task Execution & Environment	mapreduce.reduce.shuffle.input.buffer.percent	Empty implementation
Task Execution & Environment	mapreduce.reduce.input.buffer.percent	Empty implementation
Task Execution & Environment	mapreduce.job.id	Empty implementation
Task Execution & Environment	mapreduce.job.jar	Empty implementation
Task Execution & Environment	mapreduce.job.local.dir	Empty implementation
Task Execution & Environment	mapreduce.task.id	Empty implementation
Task Execution & Environment	mapreduce.task.attempt.id	Empty implementation
Task Execution & Environment	mapreduce.task.is.map	Empty implementation
Task Execution & Environment	mapreduce.task.partition	Empty implementation

Type	Interface	Is it compatible?
Task Execution & Environment	mapreduce.map.input.file	Empty implementation
Task Execution & Environment	mapreduce.map.input.start	Empty implementation
Task Execution & Environment	mapreduce.map.input.length	Empty implementation
Task Execution & Environment	mapreduce.task.output.dir	Empty implementation
JobClient	cancelDelegationToken(Token <DelegationTokenIdentifier> token)	No, throw exception
JobClient	close()	Empty implementation
JobClient	displayTasks(JobID jobId, String type, String state)	No, throw exception
JobClient	getAllJobs()	No, throw exception
JobClient	getCleanupTaskReports(JobID jobId)	No, throw exception
JobClient	getClusterStatus()	No, throw exception
JobClient	getClusterStatus(boolean detailed)	No, throw exception
JobClient	getDefaultMaps()	No, throw exception
JobClient	getDefaultReduces()	No, throw exception
JobClient	getDelegationToken(Text renewer)	No, throw exception
JobClient	getFs()	No, throw exception

Type	Interface	Is it compatible?
JobClient	getJob(JobID jobId)	No, throw exception
JobClient	getJob(String jobId)	No, throw exception
JobClient	getJobsFromQueue(String queueName)	No, throw exception
JobClient	getMapTaskReports(JobID jobId)	No, throw exception
JobClient	getMapTaskReports(String jobId)	No, throw exception
JobClient	getQueueAclsForCurrentUser()	No, throw exception
JobClient	getQueueInfo(String queueName)	No, throw exception
JobClient	getQueues()	No, throw exception
JobClient	getReduceTaskReports(JobID jobId)	No, throw exception
JobClient	getReduceTaskReports(String jobId)	No, throw exception
JobClient	getSetupTaskReports(JobID jobId)	No, throw exception
JobClient	getStagingAreaDir()	No, throw exception
JobClient	getSystemDir()	No, throw exception
JobClient	getTaskOutputFilter()	No, throw exception
JobClient	getTaskOutputFilter(JobConf job)	No, throw exception
JobClient	init(JobConf conf)	No, throw exception
JobClient	isJobDirValid(Path jobDirPath, FileSystem fs)	No, throw exception

Type	Interface	Is it compatible?
JobClient	jobsToComplete()	No, throw exception
JobClient	monitorAndPrintJob(JobConf conf, RunningJob job)	No, throw exception
JobClient	renewDelegationToken(Token<DelegationTokenIdentifier> token)	No, throw exception
JobClient	run(String[] argv)	No, throw exception
JobClient	runJob(JobConf job)	Yes
JobClient	setTaskOutputFilter(JobClient.TaskStatusFilter newValue)	No, throw exception
JobClient	setTaskOutputFilter(JobConf job, JobClient.TaskStatusFilter newValue)	No, throw exception
JobClient	submitJob(JobConf job)	Yes
JobClient	submitJob(String jobFile)	No, throw exception
JobConf	deleteLocalFiles()	No, throw exception
Jobconf	deleteLocalFiles(String subdir)	No, throw exception
Jobconf	normalizeMemoryConfigValue(long val)	Empty implementation
Jobconf	setCombinerClass(Class<? extends Reducer> theClass)	Yes
Jobconf	setCompressMapOutput(boolean compress)	Empty implementation
Jobconf	setInputFormat(Class<? extends InputFormat> theClass)	No, throw exception
JobConf	setJar(String jar)	No, throw exception
JobConf	setJarByClass(Class cls)	No, throw exception

Type	Interface	Is it compatible?
JobConf	setJobEndNotificationURI(String uri)	No, throw exception
JobConf	setJobName(String name)	Empty implementation
JobConf	setJobPriority(JobPriority prio)	No, throw exception
JobConf	setKeepFailedTaskFiles(boolean keep)	No, throw exception
JobConf	setKeepTaskFilesPattern(String pattern)	No, throw exception
JobConf	setKeyFieldComparatorOptions(String keySpec)	No, throw exception
JobConf	setKeyFieldPartitionerOptions(String keySpec)	No, throw exception
JobConf	setMapDebugScript(String mDbgScript)	Empty implementation
JobConf	setMapOutputCompressorClass(Class<? extends CompressionCodec> codecClass)	Empty implementation
JobConf	setMapOutputKeyClass(Class<? > theClass)	Yes
JobConf	setMapOutputValueClass(Class<? > theClass)	Yes
JobConf	setMapperClass(Class<? extends Mapper> theClass)	Yes
JobConf	setMapRunnerClass(Class<? extends MapRunnable> theClass)	No, throw exception
JobConf	setMapSpeculativeExecution(boolean speculativeExecution)	Empty implementation
JobConf	setMaxMapAttempts(int n)	Empty implementation
JobConf	setMaxMapTaskFailuresPercent(int percent)	Empty implementation

Type	Interface	Is it compatible?
JobConf	setMaxPhysicalMemoryForTask(long mem)	Empty implementation
JobConf	setMaxReduceAttempts(int n)	Empty implementation
JobConf	setMaxReduceTaskFailuresPercent(int percent)	Empty implementation
JobConf	setMaxTaskFailuresPerTracker(int noFailures)	Empty implementation
JobConf	setMaxVirtualMemoryForTask(long vmem)	Empty implementation
JobConf	setMemoryForMapTask(long mem)	Yes
JobConf	setMemoryForReduceTask(long mem)	Yes
JobConf	setNumMapTasks(int n)	Yes
JobConf	setNumReduceTasks(int n)	Yes
JobConf	setNumTasksToExecutePerJvm(int numTasks)	Empty implementation
JobConf	setOutputCommitter(Class<? extends OutputCommitter> theClass)	No, throw exception
JobConf	setOutputFormat(Class<? extends OutputFormat> theClass)	Empty implementation
JobConf	setOutputKeyClass(Class<? > theClass)	Yes
JobConf	setOutputKeyComparatorClass(Class<? extends RawComparator> theClass)	No, throw exception
JobConf	setOutputValueClass(Class<? > theClass)	Yes
JobConf	setOutputValueGroupingComparator(Class<? extends RawComparator> theClass)	No, throw exception
JobConf	setPartitionerClass(Class<? extends Partitioner> theClass)	Yes

Type	Interface	Is it compatible?
JobConf	setProfileEnabled(boolean newValue)	Empty implementation
JobConf	setProfileParams(String value)	Empty implementation
JobConf	setProfileTaskRange(boolean isMap, String newValue)	Empty implementation
JobConf	setQueueName(String queueName)	No, throw exception
JobConf	setReduceDebugScript(String rDbgScript)	Empty implementation
JobConf	setReducerClass(Class<? extends Reducer> theClass)	Yes
JobConf	setReduceSpeculativeExecution(boolean speculativeExecution)	Empty implementation
JobConf	setSessionId(String sessionId)	Empty implementation
JobConf	setSpeculativeExecution(boolean speculativeExecution)	No, throw exception
JobConf	setUseNewMapper(boolean flag)	Yes
JobConf	setUseNewReducer(boolean flag)	Yes
JobConf	setUser(String user)	Empty implementation
JobConf	setWorkingDirectory(Path dir)	Empty implementation
FileInputFormat	Not involved	No, throw exception

Type	Interface	Is it compatible?
TextInputFormat	Not involved	Yes
InputSplit	mapred.min.split.size.	No, throw exception
FileSplit	map.input.file	No, throw exception
RecordWriter	Not involved	No, throw exception
RecordReader	Not involved	No, throw exception
OutputFormat	Not involved	No, throw exception
OutputCommitter	abortJob(JobContext jobContext, int status)	No, throw exception
OutputCommitter	abortJob(JobContext context, JobStatus.State runState)	No, throw exception
OutputCommitter	abortTask(TaskAttemptContext taskContext)	No, throw exception
OutputCommitter	abortTask(TaskAttemptContext taskContext)	No, throw exception
OutputCommitter	cleanupJob(JobContext jobContext)	No, throw exception
OutputCommitter	cleanupJob(JobContext context)	No, throw exception
OutputCommitter	commitJob(JobContext jobContext)	No, throw exception
OutputCommitter	commitJob(JobContext context)	No, throw exception
OutputCommitter	commitTask(TaskAttemptContext taskContext)	No, throw exception
OutputCommitter	needsTaskCommit(TaskAttemptContext taskContext)	No, throw exception
OutputCommitter	needsTaskCommit(TaskAttemptContext taskContext)	No, throw exception

Type	Interface	Is it compatible?
OutputCommitter	setupJob(JobContext jobContext)	No, throw exception
OutputCommitter	setupJob(JobContext jobContext)	No, throw exception
OutputCommitter	setupTask(TaskAttemptContext taskContext)	No, throw exception
OutputCommitter	setupTask(TaskAttemptContext taskContext)	No, throw exception
Counter	getDisplayName()	Yes
Counter	getName()	Yes
Counter	getValue()	Yes
Counter	increment(long incr)	Yes
Counter	setValue(long value)	Yes
Counter	setDisplayNames(String displayName)	Yes
DistributedCache	CACHE_ARCHIVES	No, throw exception
DistributedCache	CACHE_ARCHIVES_SIZES	No, throw exception
DistributedCache	CACHE_ARCHIVES_TIMESTAMPS	No, throw exception
Distributed cache	CACHE_FILES	No, throw exception
DistributedCache	CACHE_FILES_SIZES	No, throw exception
DistributedCache	CACHE_FILES_TIMESTAMPS	No, throw exception
DistributedCache	CACHE_LOCALARCHIVES	No, throw exception
DistributedCache	CACHE_LOCALFILES	No, throw exception
DistributedCache	CACHE_SYMLINK	No, throw exception

Type	Interface	Is it compatible?
DistributedCache	addArchiveToClassPath(Path archive, Configuration conf)	No, throw exception
DistributedCache	addArchiveToClassPath(Path archive, Configuration conf, FileSystem fs)	No, throw exception
DistributedCache	addCacheArchive(Uri uri, Configuration conf)	No, throw exception
DistributedCache	addCacheFile(Uri uri, Configuration conf)	No, throw exception
DistributedCache	addFileToClassPath(Path file, Configuration conf)	No, throw exception
DistributedCache	addFileToClassPath(Path file, Configuration conf, FileSystem fs)	No, throw exception
DistributedCache	addLocalArchives(Configuration conf, String str)	No, throw exception
DistributedCache	addLocalFiles(Configuration conf, String str)	No, throw exception
DistributedCache	checkURIs(Uri[] uriFiles, Uri[] uriArchives)	No, throw exception
DistributedCache	createAllSymlink(Configuration conf, File jobCacheDir, File workDir)	No, throw exception
DistributedCache	createSymlink(Configuration conf)	No, throw exception
DistributedCache	getArchiveClassPaths(Configuration conf)	No, throw exception
DistributedCache	getArchiveTimestamps(Configuration conf)	No, throw exception
DistributedCache	getCacheArchives(Configuration conf)	No, throw exception
DistributedCache	getCacheFiles(Configuration conf)	No, throw exception
DistributedCache	getFileClassPaths(Configuration conf)	No, throw exception
DistributedCache	getFileStatus(Configuration conf, Uri cache)	No, throw exception

Type	Interface	Is it compatible?
DistributedCache	getFileTimestamps(Configuration conf)	No, throw exception
DistributedCache	getLocalCacheArchives(Configuration conf)	No, throw exception
DistributedCache	getLocalCacheFiles(Configuration conf)	No, throw exception
DistributedCache	getSymlink(Configuration conf)	No, throw exception
DistributedCache	getTimestamp(Configuration conf, URI cache)	No, throw exception
DistributedCache	setArchiveTimestamps(Configuration conf, String timestamps)	No, throw exception
DistributedCache	setCacheArchives(URI[] archives, Configuration conf)	No, throw exception
DistributedCache	setCacheFiles(URI[] files, Configuration conf)	No, throw exception
DistributedCache	setFileTimestamps(Configuration conf, String timestamps)	No, throw exception
DistributedCache	setLocalArchives(Configuration conf, String str)	No, throw exception
DistributedCache	setLocalFiles(Configuration conf, String str)	No, throw exception
IsolationRunner	Not involved	No, throw exception
Profiling	Not involved	Empty implementation
Debugging	Not involved	Empty implementation
Data Compression	Not involved	Yes
Skipping Bad Records	Not involved	No, throw exception
Job Authorization	mapred.acls.enabled	No, throw exception

Type	Interface	Is it compatible?
Job Authorization	mapreduce.job.acl-view-job	No, throw exception
Job Authorization	mapreduce.job.acl-modify-job	No, throw exception
Job Authorization	mapreduce.cluster.administrators	No, throw exception
Job Authorization	mapred.queue.queue-name.acl-administer-jobs	No, throw exception
MultipleInputs	Not involved	No, throw exception
MultipleOutputs	Not involved	Yes
org.apache.hadoop.mapreduce.lib.db	Not involved	No, throw exception
org.apache.hadoop.mapreduce.security	Not involved	No, throw exception
org.apache.hadoop.mapreduce.lib.jobcontrol	Not involved	No, throw exception
org.apache.hadoop.mapreduce.lib.chain	Not involved	No, throw exception
org.apache.hadoop.mapreduce.lib.db	Not involved	No, throw exception

4.4.3. Java Sandbox

MaxCompute, MapReduce and UDF are limited by the Java sandbox when running in the distributed environment. However, the main program of MapReduce jobs, such as MR Main, is not restricted. The specific limits are as follows.

- Direct access to local files is not allowed. You can only access files by using interfaces provided by MaxCompute MapReduce/Graph.
 - Read resources specified by the resources option, including files, jar packages, and resource tables.
 - Output log information through System.out and System.err. You can view log information by running the Log command on the MaxCompute console.
- Direct access to the distributed file system is not allowed. You can only access table records

by using MaxCompute MapReduce/Graph.

- JNI call restrictions are not allowed.
- Creation of Java threads is not allowed. Initiation of sub-processes to run Linux commands is not allowed.
- Network access, including obtaining local IP addresses, is not allowed.
- Java reflection is restricted: suppressAccessChecks permission is denied. A private attribute or method cannot be set to accessible for obtaining private attributes or calling private methods.

Specifically for the user code, access denied is thrown if you follow these steps.

- java.io.File

```
public boolean delete()
public void deleteOnExit()
public boolean exists()
public boolean canRead()
public boolean isFile()
public boolean isDirectory()
public boolean isHidden()
public long lastModified()
public long length()
public String[] list()
public String[] list(FilenameFilter filter)
public File[] listFiles()
public File[] listFiles(FilenameFilter filter)
public File[] listFiles(FileFilter filter)
public boolean canWrite()
public boolean createNewFile()
public static File createTempFile(String prefix, String suffix)
public static File createTempFile(String prefix, String suffix, File directory)
public boolean mkdir()
public boolean mkdirs()
public boolean renameTo(File dest)
public boolean setLastModified(long time)
public boolean setReadOnly()
```

- java.io.RandomAccessFile

```
RandomAccessFile(String name, String mode)
RandomAccessFile(File file, String mode)
```

- java.io.FileInputStream

```
FileInputStream(FileDescriptor fdObj)
FileInputStream(String name)
FileInputStream(File file)
```

- **java.io.FileOutputStream**

```
FileOutputStream(FileDescriptor fdObj)
FileOutputStream(File file)
FileOutputStream(String name)
FileOutputStream(String name, boolean append)
```

- **java.lang.Class**

```
public ProtectionDomain getProtectionDomain()
```

- **java.lang.ClassLoader**

```
ClassLoader()
ClassLoader(ClassLoader parent)
```

- **java.lang.Runtime**

```
public Process exec(String command)
public Process exec(String command, String envp[])
public Process exec(String cmdarray[])
public Process exec(String cmdarray[], String envp[])
public void exit(int status)
public static void runFinalizersOnExit(boolean value)
public void addShutdownHook(Thread hook)
public boolean removeShutdownHook(Thread hook)
public void load(String lib)
public void loadLibrary(String lib)
```

- **java.lang.System**


```
public static void exit(int status)
public static void runFinalizersOnExit(boolean value)
public static void load(String filename)
public static void loadLibrary( String libname)
public static Properties getProperties()
public static void setProperties(Properties props)
public static String getProperty(String key) //Only some keys are allowed for file access.
public static String getProperty(String key, String def) // Only some keys are allowed for file access.
public static String setProperty(String key, String value)
public static void setIn(InputStream in)
public static void setOut(PrintStream out)
public static void setErr(PrintStream err)
public static synchronized void setSecurityManager(SecurityManager s)
```

List of keys allowed by System.getProperty is as follows:

```
java.version
java.vendor
java.vendor.url
java.class.version
os.name
os.version
os.arch
file.separator
path.separator
line.separator
java.specification.version
java.specification.vendor
java.specification.name
java.vm.specification.version
java.vm.specification.vendor
java.vm.specification.name
java.vm.version
java.vm.vendor
java.vm.name
file.encoding
user.timezone
```

- java.lang.Thread

```
Thread()
Thread(Runnable target)
Thread(String name)
Thread(Runnable target, String name)
Thread(ThreadGroup group, ...)
public final void checkAccess()
public void interrupt()
public final void suspend()
public final void resume()
public final void setPriority(int newPriority)
public final void setName(String name)
public final void setDaemon(boolean on)
public final void stop()
public final synchronized void stop(Throwable obj)
public static int enumerate(Thread tarray[])
public void setContextClassLoader(ClassLoader cl)
```

- **java.lang.ThreadGroup**

```
ThreadGroup(String name)
ThreadGroup(ThreadGroup parent, String name)
public final void checkAccess()
public int enumerate(Thread list[])
public int enumerate(Thread list[], boolean recurse)
public int enumerate(ThreadGroup list[])
public int enumerate(ThreadGroup list[], boolean recurse)
public final ThreadGroup getParent()
public final void setDaemon(boolean daemon)
public final void setMaxPriority(int pri)
public final void suspend()
public final void resume()
public final void destroy()
public final void interrupt()
public final void stop()
```

- **java.lang.reflect.AccessibleObject**

```
public static void setAccessible(...)
public void setAccessible(...)
```

- **java.net.InetAddress**

```
public String getHostName()
public static InetAddress[] getAllByName(String host)
public static InetAddress getLocalHost()
```

- **java.net.DatagramSocket**

```
public InetAddress getLocalAddress()
```

- **java.net.Socket**

```
Socket(...)
```

- **java.net.ServerSocket**

```
ServerSocket(...)
public Socket accept()
protected final void implAccept(Socket s)
public static synchronized void setSocketFactory(...)
public static synchronized void setSocketImplFactory(...)
```

- **java.net.DatagramSocket**

```
DatagramSocket(...)
public synchronized void receive(DatagramPacket p)
```

- **java.net.MulticastSocket**

```
MulticastSocket(...)
```

- **java.net.URL**

```
URL(...)
public static synchronized void setURLStreamHandlerFactory(...)
java.net.URLConnection
public static synchronized void setContentHandlerFactory(...)
public static void setFileNameMap(FileNameMap map)
```

- **java.net.HttpURLConnection**

```
public static void setFollowRedirects(boolean set)
java.net.URLClassLoader
URLClassLoader(...)
```

- **java.security.AccessControlContext**

```
public AccessControlContext(AccessControlContext acc, DomainCombiner combiner)
public DomainCombiner getDomainCombiner()
```

5.External table

5.1. Overview of External tables

MaxCompute is the core computing component of the Alibaba Cloud big data platform. It possesses powerful computing capabilities and can schedule parallel computing tasks on a large volume of nodes. It also provides a set of proven processing management mechanisms for distributed computing failover, retry, and other functions.

As the entry of distributed data processing, MaxCompute SQL provides powerful support for quick processing and storing of exabytes of offline data. With the continuous expansion of big data business, many new cases of data usage are emerging, and the MaxCompute computing frameworks are also evolving. Access to powerful computation capabilities is gradually opening to external data sources instead of to internal data with special formats.

At this stage, MaxCompute SQL faces structured data stored in the internal MaxCompute table in cfile format. For other user data outside of MaxCompute tables (including text and various types of unstructured data), you must first import the data to MaxCompute tables using various tools and then compute it. The process of data import has great limitations. For example, to process OSS data in MaxCompute, two common methods can be used:

- To download data from OSS using the OSS SDK or other tools, the data is then imported into the table through the MaxCompute tunnel.
- Write the UDF and call the OSS SDK directly within the UDF to access the OSS data.

But there are shortcomings in both of these practices:

- You must relay data outside of the MaxCompute system. If the OSS data volume is too large, you need to consider using concurrent operations to accelerate the process and you cannot make full use of MaxCompute's large-scale computing capabilities.
- The second type typically needs to apply for UDF network access, there is also a problem for developers to control the number of job concurrency and how the data is split.

This section describes the functionality of an External table, support is designed to provide the ability to process data other than existing MaxCompute tables. In this framework, you use a simple DDL statement to create an external table in MaxCompute. Then, you can associate MaxCompute tables with the external data source to provide various data access and output capabilities. After creating an external table, you can use it like a MaxCompute table (in most situations), to take full advantage of MaxCompute SQL's powerful computing functions.

 **Note** Using the external tables feature, the data of the external tables is not copied and placed on the MaxCompute for storage.

Here, *a variety of data* covers two dimensions:

A variety of data storage media: a plug-in framework can be used to connect to a wide variety of data storage media, such as OSS, OTS.

Diverse data formats: The MaxCompute table is structured data, external tables can not be limited to structured data.

- There is no structured data, such as images, audio, video files, raw bindings, and so on.
- Semi-structured data, such as CSV, TSV, and so on, implies a certain schema text file. Structured Data for a non-cfile, such as an orc/parquet file, or even hbase/OTS data.

We'll take some examples to help you gain insight into the processing of unstructured data:

- To access OSS and OTS unstructured data see accessing [OSS unstructured data](#) and [accessing OTS unstructured data](#).
- External tables access the OSS account, and in Ram customize the permissions that authorize MaxCompute to access the OSS see [OSS STS mode authorization](#).
- The unstructured framework of MaxCompute supports output of MaxCompute data directly to OSS via insert, see [Unstructured data exported to OSS](#).
- To work with data on middleware databases, see [Processing open source format data for OSS](#) handling data in a variety of open source formats.

5.2. STS authorization

This topic describes how to customize the permissions of MaxCompute to access Object Storage Service (OSS) and Tablestore in the Resource Access Management (RAM) console. MaxCompute uses RAM and Security Token Service (STS) of Alibaba Cloud to secure data access.

OSS access permission

STS authorization for OSS

You must authorize the account used to run MaxCompute jobs to access OSS data. You can grant permissions in the following methods:

- If MaxCompute and OSS are under the same Alibaba Cloud account, log on to the RAM console and [click here to perform one-click authorization](#).
- Customize authorization.
 - i. Create a role. Log on to the [RAM console](#). In the left-side navigation pane, click RAM Roles and create a role such as AliyunODPSDefaultRole or AliyunODPSRoleForOtherUser. If MaxCompute and OSS are not under the same account, log on with the OSS account for authorization.
 - ii. Modify the policy content of a role.

```
-- If MaxCompute and OSS are under the same account, configure as follows:
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}

-- If MaxCompute and OSS are not under the same account, configure as follows:
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "ID of the Alibaba Cloud account that owns MaxCompute@odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}
```

- iii. Create the AliyunODPSRolePolicy authorization policy that contains the permissions required for accessing OSS. You can also customize other permissions.

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "oss:ListBuckets",
        "oss:GetObject",
        "oss:ListObjects",
        "oss:PutObject",
        "oss:DeleteObject",
        "oss:AbortMultipartUpload",
        "oss:ListParts"
      ],
      "Resource": "**",
      "Effect": "Allow"
    }
  ]
}
```

iv. Grant the permissions in the AliyunODPSRolePolicy authorization policy to the role.

STS authorization for Tablestore

You must authorize the account used to run MaxCompute jobs to access Tablestore data. You can grant permissions in the following methods:

- If MaxCompute and Tablestore are under the same Alibaba Cloud account, log on to the RAM console and [click here to perform one-click authorization](#).
- Customize authorization.

i. Create a role.

Log on to the [RAM console](#). In the left-side navigation pane, click RAM Roles and create a role such as AliyunODPSDefaultRole or AliyunODPSRoleForOtherUser. If MaxCompute and Tablestore are not under the same account, log on with the Tablestore account for authorization.

ii. Modify the policy content of a role.

```
-- If MaxCompute and Tablestore are under the same account, configure as follows:
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}

-- If MaxCompute and Tablestore are not under the same account, configure as follows:
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "UID of the Alibaba Cloud account that owns MaxCompute@odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}
```

 **Note** You can click the avatar in the upper-right corner to go to the **Account Management** page. On the page that appears, view the UID of your Alibaba cloud account.

iii. Edit the AliyunODPSRolePolicy authorization policy for the role.


```

{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "ots:ListTable",
        "ots:DescribeTable",
        "ots:GetRow",
        "ots:PutRow",
        "ots:UpdateRow",
        "ots>DeleteRow",
        "ots:GetRange",
        "ots:BatchGetRow",
        "ots:BatchWriteRow",
        "ots:ComputeSplitPointsBySize"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
-- You can also customize other permissions.

```

- iv. Grant the permissions in the AliyunODPSRolePolicy authorization policy to the role.

5.3. Access OSS data by using the built-in extractor

This topic describes how to access Object Storage Service (OSS) data by using the built-in extractor of MaxCompute.

Before you begin, make sure that you are authorized to access unstructured OSS data. For more information, see [STS authorization for OSS](#).

After you create a foreign table in MaxCompute and associate the foreign table with OSS, you can query the foreign table to access OSS data. For more information, see [Create a foreign table](#) and [Query a foreign table](#).

 **Note** You can create, search for, query, configure, process, and analyze foreign tables by using DataWorks, which is powered by MaxCompute. For more information, see [External table](#).

Overview

To use foreign tables to access external data sources, you can use the built-in extractor of MaxCompute. For example, you can use the built-in extractor to read data stored in the standard format in [OSS](#).

 **Note** The built-in extractor of MaxCompute cannot access DATETIME data contained in text files in OSS. To access DATETIME data, define a custom extractor. For more information, see [Use a custom extractor to access DATETIME data in OSS text files](#).

Assume that a CSV file is available in [OSS](#), the endpoint is `oss-cn-shanghai-internal.aliyuncs.com`, the bucket name is `oss-odps-test`, and the file is stored in `/demo/vehicle.csv`.

Create a foreign table

Execute the following statement to create a foreign table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external
(
  vehicleId INT,
  recordId INT,
  patientId INT,
  calls INT,
  locationLatitute DOUBLE,
  locationLongtitue DOUBLE,
  recordTime STRING,
  direction STRING
)
STORED BY 'com.aliyun.odps.CsvStorageHandler'
WITH SERDEPROPERTIES (
  'odps.properties.rolearn'='acs:ram:xxxxx:role/aliyunodpsdefaultrole'
)
LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/Demo/';
```

- **STORED BY:** specifies the name of the storage handler. `com.aliyun.odps.CsvStorageHandler` is a built-in storage handler used to process CSV files. It defines how to read and write CSV files. You can specify this parameter as required. The logic of reading and writing CSV files is implemented by the system.
- **WITH SERDEPROPERTIES:** specifies properties of the foreign table.
 - `odps.properties.rolearn`: specifies the Alibaba Cloud Resource Name (ARN) of AliyunODPSDefaultRole in Resource Access Management (RAM). If you use Security Token Service (STS) to authorize MaxCompute to access OSS, this parameter is required. For more information, see [STS authorization](#). You can obtain the parameter value from [role details](#) in the RAM console.
- **LOCATION:** specifies the OSS directory to query. The system reads all files in the directory by default.

- We recommend that you use the internal same-region endpoint of OSS to avoid fees that are incurred by OSS data flows.
- We recommend that you store OSS data in the region where you activate MaxCompute. MaxCompute can only be deployed in specific regions, so cross-region data connectivity is not ensured.
- The OSS connection format is `oss://oss-cn-shanghai-internal.aliyuncs.com/Bucket name/Directory/`. Do not add a file name after the Directory parameter. The following examples are negative:

```
http://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo/ -- HTTP connections are not supported.
https://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo/ -- HTTPS connections are not supported.
oss://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo -- The connection address is invalid.
oss://oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/Demo/vehicle.csv -- You do not need to specify the file name.
```

 **Note** By creating a foreign table, you only set up a connection to the destination OSS directory. After you delete the foreign table, data stored in the OSS directory is retained.

You can execute the following statement to view the structure of the created foreign table:

```
DESC EXTENDED <table_name>;
```

In the output, you can view basic table information, which is similar to the information returned for a created internal table. In addition to basic table information, you can view the storage handler and the OSS directory to query.

Query a foreign table

After a foreign table is created, you can use it the same way that you use a common table. Assume that `/demo/vehicle.csv` contains the following data:

```

1,1,51,1,46.81006,-92.08174,9/14/2014 0:00,S
1,2,13,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,3,48,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,4,30,1,46.81006,-92.08174,9/14/2014 0:00,W
1,5,47,1,46.81006,-92.08174,9/14/2014 0:00,S
1,6,9,1,46.81006,-92.08174,9/14/2014 0:00,S
1,7,53,1,46.81006,-92.08174,9/14/2014 0:00,N
1,8,63,1,46.81006,-92.08174,9/14/2014 0:00,SW
1,9,4,1,46.81006,-92.08174,9/14/2014 0:00,NE
1,10,31,1,46.81006,-92.08174,9/14/2014 0:00,N

```

Execute the following sample statement to query data:

```
SELECT recordId, patientId, direction FROM ambulance_data_csv_external WHERE patientId > 25;
```

Note

- A foreign table can only be managed by using the MaxCompute SQL engine, not MaxCompute MapReduce.
- To obtain data by using HTTPS at the underlying layer, commit the `set odps.sql.unstructured.data.oss.use.https=true;` command with SQL statements for execution.

The preceding sample command submits a job, which calls the built-in CSV extractor to read and process data from OSS. The following result is returned:

```

+-----+-----+-----+
| recordId | patientId | direction |
+-----+-----+-----+
| 1       | 51       | S       |
| 3       | 48       | NE      |
| 4       | 30       | W       |
| 5       | 47       | S       |
| 7       | 53       | N       |
| 8       | 63       | SW      |
| 10      | 31       | N       |
+-----+-----+-----+

```

Use the MSCK REPAIR TABLE statement to add a partition to a foreign table

MaxCompute supports automatically adding partitions to foreign tables based on the OSS directory where the data is stored. Execute the following statement to add a partition to a foreign table:

```
MSCK [REPAIR] TABLE external_table_name [ADD PARTITIONS];
```

Syntax:

- To import data to OSS, make sure that the OSS directory is in the following format: `oss://xxx/table-location/ptname1=ptvalue1/ptname2=ptvalue2/xxx`.
- When you create a foreign table, you must specify the partition structure.
- After you execute the `MSCK [REPAIR] TABLE external_table_name [ADD PARTITIONS];` statement, SQL automatically parses the OSS directory to recognize the partition, and adds the partition to the foreign table.

Example:

1. Upload data to OSS. The following figure shows the OSS directory.



2. Execute the following statement to create a foreign table named `orc_pt_v0`, in which the structure of the `pt` partition is specified:

```
CREATE EXTERNAL TABLE orc_pt_v0
(
  name STRING
)
PARTITIONED BY (pt bigint)
STORED BY 'com.aliyun.odps.CsvStorageHandler'
WITH SERDEPROPERTIES (
  'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
)
LOCATION 'oss://xxx/odps-ext-reg-perf/orc-pt-v0';
```

3. Execute the following statement to add a partition to `orc_pt_v0`:

```
MSCK REPAIR TABLE orc_pt_v0 ADD PARTITIONS;
-- In this case, the MSCK statement is equivalent to the following three statements:
ALTER TABLE orc_pt_v0 ADD PARTITION (pt=1);
ALTER TABLE orc_pt_v0 ADD PARTITION (pt=10);
ALTER TABLE orc_pt_v0 ADD PARTITION (pt=100);
```

Read CSV or TSV files compressed in GZIP format

MaxCompute can read CSV and TSV files in GZIP format from OSS only by using the built-in extractor. The main difference between reading non-compressed and compressed files is the property specified by `SERDEPROPERTIES`.

 **Note** If the data file to be read is an archived object in OSS, log on to the OSS console to restore the object. For more information, see the [OSS documentation](#).

Use the following statement to create a foreign table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_csv_external
(
  vehicleId BIGINT,
  recordId BIGINT,
  patientId BIGINT,
  calls BIGINT,
  locationLatitude DOUBLE,
  locationLongitude DOUBLE,
  recordTime STRING,
  direction STRING
)
STORED BY 'com.aliyun.odps.CsvStorageHandler'
WITH SERDEPROPERTIES (
  'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
  ['odps.text.option.gzip.input.enabled'='true']
  ['name3'='value3']
)
LOCATION 'oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/Demo/SampleData/CSV/AmbulanceData/';
```

The following table describes the properties supported by SERDEPROPERTIES.

Property	Valid value	Default value	Description
odps.text.option.gzip.input.enabled	• True • False	False	Specifies whether to compress the file before data read.
odps.text.option.gzip.output.enabled	• True • False	False	Specifies whether to compress the file before data write.
odps.text.option.header.lines.count	Non-negative integers	0	Specifies the number of lines to skip in the file.

Property	Valid value	Default value	Description
<code>odps.text.option.null.indicator</code>	Strings	""	Specifies the strings in the file to be parsed as NULL in SQL. For example, if you specify <code>\N</code> to represent NULL, <code>a,\N,b</code> is parsed as <code>a, NULL, b</code> in SQL.
<code>odps.text.option.ignore.empty.lines</code>	- True - False	True	Specifies whether to ignore blank lines.
<code>odps.text.option.encoding</code>	- UTF-8 - UTF-16 - US-ASCII	UTF-8	Specifies the encoding format of the file.
<code>odps.text.option.delimiter</code>	Single characters	,	Specifies the column delimiter of the file.
<code>odps.text.option.use.quote</code>	- True - False	False	Specifies whether to recognize column delimiters in the CSV file if it uses double quotation marks (<code>"</code>) as column delimiters. Fields in the CSV file that contain a carriage return/line feed (CR/LF) pair, double quotation mark (<code>"</code>), or comma (<code>,</code>) for separating multiple values must be enclosed in double quotation marks. Note that if a field contains a double quotation mark (<code>"</code>), replace the double quotation mark (<code>"</code>) with two double quotation marks (<code>"</code>) for escaping.

 Note

If you need to read data from and write data to a foreign table associated with compressed OSS data, set both `odps.text.option.gzip.input.enabled` and `odps.text.option.gzip.output.enabled` to True to create the foreign table.

5.4. Access unstructured data in OSS by using a custom extractor

This topic describes how to access unstructured data in Object Storage Service (OSS) by using a custom extractor.

Access text data in OSS by using a custom extractor

If OSS data is in a complicated format and cannot be processed by the built-in extractor, you must customize an extractor to read the OSS data.

Assume that a TXT file that contains the following data is stored in OSS. The columns of records in the file are separated by vertical bars (|). The file path is

```
/demo/SampleData/CustomTxt/AmbulanceData/vehicle.csv .
```

```
1|1|51|1|46.81006|-92.08174|9/14/2014 0:00|S
1|2|13|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|3|48|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|4|30|1|46.81006|-92.08174|9/14/2014 0:00|W
1|5|47|1|46.81006|-92.08174|9/14/2014 0:00|S
1|6|9|1|46.81006|-92.08174|9/14/2014 0:00|S
1|7|53|1|46.81006|-92.08174|9/14/2014 0:00|N
1|8|63|1|46.81006|-92.08174|9/14/2014 0:00|SW
1|9|4|1|46.81006|-92.08174|9/14/2014 0:00|NE
1|10|31|1|46.81006|-92.08174|9/14/2014 0:00|N
```

1. Define an extractor.

Define a common extractor with a specified delimiter. The extractor can process all TXT files with data in the similar format. Example:

```
/**
 * Text extractor that extract schematized records from formatted plain-text(csv, tsv etc.)
 **/

public class TextExtractor extends Extractor {
    private InputStreamSet inputs;
    private String columnDelimiter;
    private DataAttributes attributes;
    private BufferedReader currentReader;
    private boolean firstRead = true;
    public TextExtractor() {
        // default to ",", this can be overwritten if a specific delimiter is provided (via DataAttributes)
        this.columnDelimiter = ",";
    }
    // no particular usage for execution context in this example
    @Override
    public void setup(ExecutionContext ctx, InputStreamSet inputs, DataAttributes attributes) {
        this.inputs = inputs; // inputs specifies an InputStreamSet. An InputStream is returned each time
        next() is called. This InputStream can read all the content of an OSS object.
        this.attributes = attributes;
```



```

    // check if "delimiter" attribute is supplied via SQL query
    String columnDelimiter = this.attributes.getValueByKey("delimiter"); // The delimiter can be used as a parameter in DDL statements.
    if ( columnDelimiter != null)
    {
        this.columnDelimiter = columnDelimiter;
    }
    // note: more properties can be initied from attributes if needed
}
@Override
public Record extract() throws IOException { // The extractor returns a record that is extracted from the foreign table.
    String line = readNextLine();
    if (line == null) {
        return null; // If NULL is returned, all records in the table have been read.
    }
    return textLineToRecord(line); // textLineToRecord splits a row into multiple columns by using the delimiter.
}
@Override
public void close(){
    // no-op
}
}

```

 **Note** For more information about the implementation process, see [TextExtractor.java](#).

2. Define a storage handler.

A storage handler is a unified entry to your custom logic for processing foreign tables.

```
package com.aliyun.odps.udf.example.text;

public class TextStorageHandler extends OdpsStorageHandler {

    @Override
    public Class<? extends Extractor> getExtractorClass() {
        return TextExtractor.class;
    }

    @Override
    public Class<? extends Outputter> getOutputterClass() {
        return TextOutputter.class;
    }
}
```

3. Compile and pack the code.

Compile the custom code and pack it into a JAR package. Then, upload the JAR package to MaxCompute.

```
add jar odps-udf-example.jar;
```

4. Create a foreign table.

Before you use a custom extractor, you must create a foreign table with the specified custom storage handler.

Execute the following SQL statement to create a foreign table, in which the delimiter parameter specifies the delimiter for splitting each row into columns:

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_txt_external
(
  vehicleId int,
  recordId int,
  patientId int,
  calls int,
  locationLatitude double,
  locationLongitude double,
  recordTime string,
  direction string
)
STORED BY 'com.aliyun.odps.udf.example.text.TextStorageHandler'
  with SERDEPROPERTIES (
    'delimiter'='\|',
    'odps.properties.rolearn'='acs:ram::xxxxxxxxxxxx:role/aliyunodpsdefaultrole'
  )
LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/Demo/SampleData/CustomTxt/AmbulanceData/'
USING 'odps-udf-example.jar';
```

- **STORED BY:** specifies the class name of a custom storage handler.
- **WITH SERDEPROPERTIES:** specifies the following parameters for the foreign table.
 - **delimiter:** specifies the delimiter.
 - **odps.properties.rolearn:** specifies the Alibaba Cloud Resource Name (ARN) for the AliyunODPSDefaultRole role in Resource Access Management (RAM). You can obtain the ARN from the role details page in the [RAM console](#).
- **LOCATION:** specifies an OSS folder. By default, the system reads all files in the folder.
 - We recommend that you use the internal same-region endpoint of OSS to avoid fees that are incurred by OSS data flows.
 - We recommend that you store OSS data in the region where you activate MaxCompute. MaxCompute can be deployed only in specific regions, so cross-region data connectivity is not ensured.

- The OSS connection format is `oss://oss-cn-shanghai-internal.aliyuncs.com/Bucket name/Folder name/`. Do not add a file name after the Folder name parameter. The following examples are negative:

```
http://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo/ // HTTP connections are not supported.
https://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo/ // HTTPS connections are not supported.
oss://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo // The connection address is invalid.
oss://oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/Demo/vehicle.csv // You do not need to specify the file name.
```

- USING: specifies the JAR package that defines the specified class.

5. Execute the following SQL statement to query the foreign table:

```
SELECT recordId, patientId, direction FROM ambulance_data_txt_external WHERE patientId > 25;
```

Access non-text data in OSS by using a custom extractor

This section describes how to use a custom extractor to access and process non-text data in OSS. WAV files are used in this example.

1. Execute the following SQL statement to create a foreign table. Define the schema of the foreign table for the custom extractor to extract the following information from the WAV files:
 - sentence_snr: the signal-to-noise ratio (SNR) of sentences in each WAV file.
 - id: the name of each WAV file.

```
CREATE EXTERNAL TABLE IF NOT EXISTS speech_sentence_snr_external
(
  sentence_snr double,
  id string
)
STORED BY 'com.aliyun.odps.udf.example.speech.SpeechStorageHandler'
WITH SERDEPROPERTIES (
  'mlfFileName'='sm_random_5_utterance.text.label',
  'speechSampleRateInKHz' = '16'
)
LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/dev/SpeechSentenceTest/'
USING 'odps-udf-example.jar,sm_random_5_utterance.text.label';
```

- com.aliyun.odps.udf.example.speech.SpeechStorageHandler: encapsulates an extractor to calculate the average SNR of valid sentences in each WAV file, and run SQL jobs to

process the structured data that is extracted.

- **LOCATION:** specifies the folder, `oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/dev/SpeechSentenceTest/` in this example, that stores multiple original WAV files. MaxCompute reads all the files and allocates these files to multiple compute nodes for processing.

2. Execute the following statement to query the foreign table:

```
select sentence_snr, id from speech_sentence_snr_external where sentence_snr > 10.0;
```

The following data is returned:

```
-----
| sentence_snr |          id          |
-----
| 34.4703 | J310209090013_H02_K03_042 |
-----
| 31.3905 | tsh148_seg_2_3013_3_6_48_80bd359827e24dd7_0 |
-----
| 35.4774 | tsh148_seg_3013_1_31_11_9d7c87aef9f3e559_0 |
-----
| 16.0462 | tsh148_seg_3013_2_29_49_f4cb0990a6b4060c_0 |
-----
| 14.5568 | tsh_148_3013_5_13_47_3d5008d792408f81_0 |
-----
```

A custom extractor allows you to execute SQL statements to process multiple audio files in OSS in a distributed manner. You can process other unstructured data such as images and videos in the same way to make full use of the powerful computing capabilities of MaxCompute.

Considerations

A custom extractor that is defined by using the method described in this topic cannot access data of the DATETIME type in OSS text files. For more information about how to define a custom extractor to access data of the DATETIME type in OSS text files, see [Access data of the DATETIME type in OSS text files by using a MaxCompute custom extractor](#).

5.5. Partition data by using foreign tables

This topic describes how to partition data by using foreign tables.

data partition

You can use the LOCATION parameter to specify an Object Storage Service (OSS) folder that maps a foreign table. After you specify an OSS folder by using the LOCATION parameter, MaxCompute reads all data in the folder, including all files in the sub-folders. If the OSS folder contains a large amount of data or data in the OSS folder increases over time, a full-text scan will cause extra I/O operations and processing time. You can use one of the following methods to address this issue:

- Store data in multiple OSS folders that map different foreign tables: You can specify multiple

OSS folders to store data and use the `LOCATION` parameter to specify each of the folders for a foreign table. Then, MaxCompute can read data in each folder by using the corresponding foreign table.

- Store data in an OSS folder that maps a foreign partitioned table: MaxCompute allows you to use foreign partitioned tables to better manage data.

Standard organization method and path structure of partitioned data in OSS

MaxCompute does not have the permission to manage data stored in external storage such as OSS. If MaxCompute needs to access OSS data by using a foreign partitioned table, make sure that the storage path of the OSS data is in the following format:

```
partitionKey1=value1\partitionKey2=value2\...
```

1. Create a foreign partitioned table.

Assume that your daily log files are stored in OSS, and you want to use MaxCompute to process the data on a daily basis. If the log files are in the CSV format and can be accessed by using the built-in extractor or a custom extractor, you can execute the following SQL statement to create a foreign partitioned table:

```
CREATE EXTERNAL TABLE log_table_external (  
    click STRING,  
    ip STRING,  
    url STRING  
)  
PARTITIONED BY (  
    year STRING,  
    month STRING,  
    day STRING  
)  
STORED BY 'com.aliyun.odps.CsvStorageHandler'  
WITH SERDEPROPERTIES (  
    'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'  
)  
LOCATION 'oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/log_data/';
```

You can use the `PARTITIONED BY` clause to specify the foreign table as a foreign partitioned table. In this example, the partitioned table has three levels of partitions with the partition keys `year`, `month`, and `day`, respectively.

For the partitions to take effect, you must specify an OSS folder in the same format as the folder specified by `LOCATION` in the preceding example. The following example shows a valid folder:

```

osscmd ls oss://oss-odps-test/log_data/
2017-01-14 08:03:35 128MB Standard oss://oss-odps-test/log_data/year=2016/month=06/day=01/
logfile
2017-01-14 08:04:12 127MB Standard oss://oss-odps-test/log_data/year=2016/month=06/day=01/
logfile.1
2017-01-14 08:05:02 118MB Standard oss://oss-odps-test/log_data/year=2016/month=06/day=02/
logfile
2017-01-14 08:06:45 123MB Standard oss://oss-odps-test/log_data/year=2016/month=07/day=10/
logfile
2017-01-14 08:07:11 115MB Standard oss://oss-odps-test/log_data/year=2016/month=08/day=08/
logfile
...

```

 **Note** In this example, data is uploaded to OSS by using OSS tools such as `osscmd`. Therefore, the storage path of the data is the path to which the data is uploaded.

2. Execute `ALTER TABLE ADD PARTITION` statements to add partitions to the foreign partitioned table.

```

ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '06', day = '01')
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '06', day = '02')
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '07', day = '10')
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '08', day = '08')
...

```

 **Note** The preceding operations are the same as those on an internal partitioned table in MaxCompute. For more information about partitions, see [Partition](#). If you have prepared data and added partitions to a foreign table of MaxCompute, you can execute SQL statements to process the OSS data that maps these partitions.

3. Analyze data.

- Execute the following SQL statement to obtain the number of distinctive IP addresses in the logs that were recorded on June 1, 2016:

```

SELECT count(distinct(ip)) FROM log_table_external WHERE year = '2016' AND month = '06' AND d
ay = '01';

```

MaxCompute accesses only the files, *logfile* and *logfile.1* in this example, in the *log_data/year=2016/month=06/day=01* sub-folder but not all files in the *log_data* folder. This prevents unnecessary I/O operations.

- Execute the following SQL statement to obtain the number of distinctive IP addresses in the logs that were recorded in the second half of 2016:

```

SELECT count(distinct(ip)) FROM log_table_external WHERE year = '2016' AND month > '06';

```

Custom path of partitioned data in OSS

If you have historical data stored in OSS paths that are not in the

`partitionKey1=value1\partitionKey2=value2\...` format, you can add partitions that map these OSS paths to a foreign table of MaxCompute. This way, MaxCompute can access data in these OSS paths.

The following example shows an OSS path that contains only partition values but not partition keys:

```
osscmd ls oss://oss-odps-test/log_data_customized/
2017-01-14 08:03:35 128MB Standard oss://oss-odps-test/log_data_customized/2016/06/01/logfile
2017-01-14 08:04:12 127MB Standard oss://oss-odps-test/log_data_customized/2016/06/01/logfile.1
2017-01-14 08:05:02 118MB Standard oss://oss-odps-test/log_data_customized/2016/06/02/logfile
2017-01-14 08:06:45 123MB Standard oss://oss-odps-test/log_data_customized/2016/07/10/logfile
2017-01-14 08:07:11 115MB Standard oss://oss-odps-test/log_data_customized/2016/08/08/logfile
...
```

You can execute the following SQL statement to map each sub-folder to a specific partition:

```
ALTER TABLE log_table_external ADD PARTITION (year = '2016', month = '06', day = '01')
LOCATION 'oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/log_data_customized/2016/06/01/';
```

In the preceding SQL statement, the `LOCATION` parameter specifies an OSS path. This path maps the partition that is added to the foreign table by using the `ADD PARTITION` clause. This way, MaxCompute can access data in the OSS path even if the path is not in the `partitionKey1=value1\partitionKey2=value2\...` format.

5.6. Process OSS data stored in open-source formats

This topic describes how to use the MaxCompute unstructured framework to process Object Storage Service (OSS) data that is stored in open-source formats. The formats include ORC, PARQUET, SEQUENCEFILE, RCFILE, AVRO, and TEXTFILE.

process OSS data stored in open-source formats

You can create, search, configure, and process external tables in the DataWorks console. You can also query and analyze data in external tables. For more information, see [External table](#).

For data that is stored in open-source formats, the unstructured framework calls the implementation method provided by an open-source community to parse the data. The method can be seamlessly integrated with MaxCompute to read and parse data stored in open-source formats.

 **Note** Before you parse the OSS data that is stored in open-source formats, perform **STS authorization** on OSS.

Syntax for creating an external table

The MaxCompute unstructured framework uses external tables to access data stored in various formats. The following code describes the syntax that is used to create an external table. Then, the table can be used to access OSS data that is stored in open-source formats.

```
DROP TABLE [IF EXISTS] <external_table>;
CREATE EXTERNAL TABLE [IF NOT EXISTS] <external_table>
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
[ROW FORMAT SERDE '<serde class>']
[WITH SERDEPROPERTIES ('odps.properties.rolearn'='${roleran}' [, 'name2'='value2',...])]
]
STORED AS <file format>
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

The syntax is similar to that in Hive. The syntax is described as follows:

- **column schemas:** defines the columns of the external table. The column definition must be the same as the definition of OSS data.
- **ROW FORMAT SERDE:** This clause is required only when you use special formats, such as `TEXTFILE`.
- **WITH SERDEPROPERTIES:** If **STS authorization** is performed on OSS, this clause is required to set the `odps.properties.rolearn` property. The property is the Alibaba Cloud Resource Name (ARN) of the RAM role. You can configure `STORED AS <file format>` and `<serde class>` to specify the file format. The ORC format is used in the following example:

```
CREATE EXTERNAL TABLE [IF NOT EXISTS] <external_table>
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
ROW FORMAT SERDE 'org.apache.hadoop.hive ql.io.orc.OrcSerde'
WITH SERDEPROPERTIES ('odps.properties.rolearn'='${roleran}')
STORED AS ORC
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/'
```

 **Note** If you do not perform STS authorization, you do not need to set the `odps.properties.rolearn` property. You only need to specify the `AccessKey ID` and `AccessKey secret` for the Location clause in plaintext.

- **STORED AS:** It is unique for reading data that is stored in open-source formats. It is different from the `STORED BY` clause that is used to **create a standard unstructured external table**. `STORED AS` is followed by the file format, such as `ORC`, `PARQUET`, `RCFILE`, `SEQUENCEFILE`, or `TEXTFILE`. In the `STORED AS` clause, the size of a file cannot be greater than 3 GB. If the size of a file exceeds 3 GB, split the file.

The mappings between file formats and SerDe classes are as follows:

- SEQUENCEFILE: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
- TEXTFILE: org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
- RCFILE: org.apache.hadoop.hive.serde2.columnar.LazyBinaryColumnarSerDe
- ORC: org.apache.hadoop.hive ql.io.orc.OrcSerde
- ORCFILE: org.apache.hadoop.hive ql.io.orc.OrcSerde
- PARQUET: org.apache.hadoop.hive ql.io.parquet.serde.ParquetHiveSerDe
- AVRO: org.apache.hadoop.hive.serde2.avro.AvroSerDe
- LOCATION: If the OSS service is associated, configure the AccessKey ID and AccessKey secret in plaintext. Example:

```
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@${endpoint}/${bucket}/${userPath}/'
```

You cannot use external tables to access OSS data by using the public endpoint of OSS.

Create an external table based on a PARQUET object in an OSS bucket

Assume that some objects are stored in an OSS directory. These objects are stored in the PARQUET format, and each object contains 16 columns: four BIGINT-type columns, four DOUBLE-type columns, and eight STRING-type columns. The DDL statement to create a table is as follows:

```
CREATE EXTERNAL TABLE tpch_lineitem_parquet
(
  l_orderkey bigint,
  l_partkey bigint,
  l_suppkey bigint,
  l_linenummer bigint,
  l_quantity double,
  l_extendedprice double,
  l_discount double,
  l_tax double,
  l_returnflag string,
  l_linestatus string,
  l_shipdate string,
  l_commitdate string,
  l_receiptdate string,
  l_shipinstruct string,
  l_shipmode string,
  l_comment string
)
STORED AS PARQUET
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/parquet_data/';
```

Data stored in the PARQUET format is not compressed by default. If you want to compress such data in MaxCompute, add the `set odps.sql.hive.compatible=true;` flag. The format of data after compression depends on the value of the TBLPROPERTIES parameter.

- `'mcfed.parquet.compression'='SNAPPY'` : The format is *SNAPPY*.
- `'mcfed.parquet.compression'='GZIP'` : The format is *GZIP*.

Create an external table based on a TEXTFILE object in an OSS bucket

We recommend that you use MaxCompute partitioned tables to access data. This applies if data of the JSON type is saved as a TEXTFILE object and TEXTFILE objects are stored in different OSS directories. These objects are stored by using the same storage and naming methods. The DDL statement to create a partitioned table is as follows:

```
CREATE EXTERNAL TABLE tpch_lineitem_textfile
(
  l_orderkey bigint,
  l_partkey bigint,
  l_suppkey bigint,
  l_linenummer bigint,
  l_quantity double,
  l_extendedprice double,
  l_discount double,
  l_tax double,
  l_returnflag string,
  l_linestatus string,
  l_shipdate string,
  l_commitdate string,
  l_receiptdate string,
  l_shipinstruct string,
  l_shipmode string,
  l_comment string
)
PARTITIONED BY (ds string)
ROW FORMAT serde 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/text_data/';
```

The subdirectories under the OSS table directory are organized by partition name. Example:

```
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/text_data/ds=20170102/'
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/text_data/ds=20170103/'
...
```

In this case, execute the following statements to add partitions:

```
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170102");
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170103");
```

The subdirectories under the OSS table directory are not organized by partition name or do not exist under the OSS table directory. Example:

```
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/text_data_20170102/;
oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/text_data_20170103/;
...
```

In this case, execute the following statements to add partitions:

```
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170102")
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/text_data_20170102/';
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170103")
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.aliyuncs.com/bucket/text_data_20170103/';
...
```

The following code describes the default value of the ROW_FORMAT clause in the DDL statement:

```
Fields terminator:'\001'
Escape delimiter:'\'
Collection items terminator:'\002'
Map keys terminator:'\003'
Lines terminator:'\n'
Null defination:'\N'
```

Create an external table based on a CSV object in an OSS bucket

The DDL statement to create an external table is as follows:

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES
  ('separatorChar','=', 'quoteChar'='', 'escapeChar'='\')
STORED AS TEXTFILE
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}';
```

The following code describes the fields and their default values for the WITH SERDEPROPERTIES clause in the DDL statement:

```
separatorChar:','
quoteChar:''
escapeChar:'\'
```

 **Note** Hive OpenCSVSerde supports only the STRING type.

OpenCSVSerde is not a built-in SerDe. To execute the DML statement, you must add the `set odps.sql.hive.compatible=true;` flag.

Create an external table based on a JSON object in an OSS bucket

The DDL statement to create an external table supports the WITH SERDEPROPERTIES clause. The DDL statement is as follows:

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe'
STORED AS TEXTFILE
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}';
```

Create an external table based on an ORC object in an OSS bucket

The DDL statement to create an external table is as follows:

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED AS ORC
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}';
```

Create an external table based on an AVRO object in an OSS bucket

The DDL statement to create an external table is as follows:

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED AS AVRO
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

Create an external table based on a SEQUENCEFILE object in an OSS bucket

The DDL statement to create an external table is as follows:

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED AS SEQUENCEFILE
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

Read and process OSS data stored in open-source formats

The preceding DDL statements show that you only need to modify the value after **STORED AS** to create external tables for different object formats. This section describes how to use the `tpch_lineitem_parquet` external table that is created based on a **PARQUET** object. To use external tables that are created based on objects of different formats, you only need to set **STORED AS** to **PARQUET**, **ORC**, **TEXTFILE**, or **RCFILE**.

- Read and process OSS data stored in open-source formats

After you create an external table and associate it with specific data, you can manage the created table as a standard MaxCompute table.

```
SELECT l_returnflag, l_linestatus,
SUM(l_extendedprice*(1-l_discount)) AS sum_disc_price,
AVG(l_quantity) AS avg_qty,
COUNT(*) AS count_order
FROM tpch_lineitem_parquet
WHERE l_shipdate <= '1998-09-02'
GROUP BY l_returnflag, l_linestatus;
```

`tpch_lineitem_parquet` is used as an internal table. However, the MaxCompute internal computing engine directly reads **PARQUET** data from OSS for processing.

`odps.sql.hive.compatible` is set to `false` if you use the `ROW FORMAT` and `STORED AS` clauses to create the `tpch_lineitem_textfile` external partitioned table. To properly read data, you must set `set odps.sql.hive.compatible` to `true`. Otherwise, an error is returned.

```
SELECT * FROM tpch_lineitem_textfile LIMIT 1;
FAILED: ODPS-0123131:User defined function exception - Traceback:
com.aliyun.odps.udf.UDFException: java.lang.ClassNotFoundException: com.aliyun.odps.hive.wrapper.HiveStorageHandlerWrapper
-- Add the following flag that is compatible with Hive:
set odps.sql.hive.compatible=true;
SELECT * FROM tpch_lineitem_textfile LIMIT 1;
```

l_orderkey	l_partkey	l_suppkey	l_linenum	l_quantity	l_extendedprice	l_discount	l_tax	l_returnflag	l_linestatus	l_shipdate	l_commitdate	l_receiptdate	l_shipinstruct	l_shipmode	l_comment
5640000001	174458698	9458733	1	14.0	23071.58	0.08	0.06	N	O	1998-01-26	1997-11-16	1998-02-18	TAKE BACK RETURN	SHIP	cuses nag silently. quick

 **Note** If you use an external table to read data, each data read operation triggers I/O operations on OSS data, and MaxCompute performance optimization for internal storage cannot take effect. As a result, the data reading performance may deteriorate. If you require repeated data computing or high computing efficiency, we recommend that you import data into MaxCompute for computing.

If complex data types are used in SQL statements, such as CREATE, SELECT, and INSERT, you must add the `set odps.sql.type.system.odps2=true;` flag before the statements. Then, commit the statements for execution. For more information, see [Date types](#).

- Import data stored in open-source formats into MaxCompute for computing

Create a MaxCompute internal table named `tpch_lineitem_internal` that has the same schema as the external table. Import OSS data stored in open-source formats into the newly created internal table. Save the data in the internal storage format.

```
CREATE TABLE tpch_lineitem_internal LIKE tpch_lineitem_parquet;
INSERT OVERWRITE TABLE tpch_lineitem_internal;
SELECT * FROM tpch_lineitem_parquet;
```

Execute the query statement that is executed on the external table. This improves computing performance.

```
SELECT l_returnflag, l_linestatus,  
SUM(l_extendedprice*(1-l_discount)) AS sum_disc_price,  
AVG(l_quantity) AS avg_qty,  
COUNT(*) AS count_order  
FROM tpch_lineitem_internal  
WHERE l_shipdate <= '1998-09-02'  
GROUP BY l_returnflag, l_linestatus;
```

FAQ

Error: Inline data exceeds the maximum allowed size .

Cause: OSS storage limits the size of each object. If the size of an object exceeds 3 GB, an error is reported.

Solution: Adjust the values of the following flags. The flags control the volume of data that each reducer can write into OSS storage. You can change the values of the flags to adjust execution plans.

```
set odps.sql.mapper.split.size=256; # Adjust the volume of table data that is read by each mapper. Unit: MB.  
set odps.sql.reducer.instances=100; # Adjust the number of reducers in the execution plan.
```

5.7. Export unstructured data to OSS

The unstructured data processing framework of MaxCompute allows you to export MaxCompute data to OSS through the INSERT method. MaxCompute also associates OSS with external tables to export data to OSS.

You can create, search for, configure, and process external tables through a visual interface in the DataWorks console. You can also query, compute, and analyze data in external tables. DataWorks is powered by MaxCompute. For more information, see [External table](#).

[Access OSS unstructured data](#) describes how MaxCompute can access and process unstructured data stored in OSS through external tables.

The two scenarios for exporting data to OSS are as follows:

- Data in MaxCompute internal tables is exported to external tables that are associated with OSS.
- After MaxCompute processes data stored in OSS through external tables, the results are directly exported to the external tables.

Similar to how you access OSS data, you can export data to OSS through a built-in or custom StorageHandler in MaxCompute.

Export data to OSS through a built-in StorageHandler

Using a built-in StorageHandler in MaxCompute, you can export data to OSS in the specified format. You only need to create an external table and specify the built-in StorageHandler to associate OSS with the table. The system will then implement the related logic.

MaxCompute supports two built-in StorageHandlers:

- `com.aliyun.odps.CsvStorageHandler` defines how to read and write data in CSV format. With this specification, the column delimiter is a comma (,) and the line break is `\n`.
- `com.aliyun.odps.TsvStorageHandler` defines how to read and write data in TSV format. With this specification, the column delimiter is `\t` and the line break is `\n`.
- Create an external table

```
CREATE EXTERNAL TABLE [IF NOT EXISTS] <external_table>
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED BY '<StorageHandler>'
[WITH SERDEPROPERTIES ( 'odps.properties.rolearn'='${roleran}') ]
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

- The `<StorageHandler>` parameter following `STORED BY` specifies the built-in StorageHandler corresponding to CSV or TSV files. If the data file to be exported to OSS is a TSV file, use `com.aliyun.odps.TsvStorageHandler`. If the data file to be exported to OSS is a CSV file, use `com.aliyun.odps.CsvStorageHandler`.
- When granting OSS data access permissions through Custom Authorization of STS Mode Authorization, you must use `WITH SERDEPROPERTIES` to specify the `odps.properties.rolearn` property. The property value is the Alibaba Cloud Resource Name (ARN) of a custom role used in RAM.

 **Note** For more information about STS mode authorization, see [Access OSS unstructured data](#).

- `LOCATION` specifies the file storage path in OSS. If the `odps.properties.rolearn` property is not set in the `WITH SERDEPROPERTIES` clause and a plaintext AccessKey pair is used for authorization, `LOCATION` is set as follows:

```
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@${endpoint}/${bucket}/${userPath}/'
```

- Perform an INSERT operation on the external table to export data to OSS.

 **Note** The size of a single file inserted into OSS cannot exceed 5 GB.

After the external table is associated with the OSS storage path, you can perform standard SQL INSERT OVERWRITE/INSERT INTO operations on the external table to export data to OSS.

```
INSERT OVERWRITE|INTO TABLE <external_tablename> [PARTITION (partcol1=val1, partcol2=val2 ...)]
select_statement
FROM <from_tablename>
[WHERE where_condition];
```

- *from_tablename*: This table can either be an internal or external table, including an external table stored in OSS or Table Store.

- INSERT writes external table data to OSS based on the StorageHandler format (TSV or CSV) specified by the STORED BY clause.

After the INSERT operation is complete, a series of files will be generated in the corresponding OSS storage path specified by LOCATION.

For example, if the OSS storage location of the external table is *oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/tsv_output_folder/*, a series of files will be generated in *tsv_output_folder*:

```
osscommand ls oss://oss-odps-test/tsv_output_folder/
2017-01-14 06:48:27 39.00B Standard oss://oss-odps-test/tsv_output_folder/.odps/.meta
2017-01-14 06:48:12 4.80MB Standard oss://oss-odps-test/tsv_output_folder/.odps/20170113224724
561g9m6csz7/M1_0_0-0.tsv
2017-01-14 06:48:05 4.78MB Standard oss://oss-odps-test/tsv_output_folder/.odps/20170113224724
561g9m6csz7/M1_1_0-0.tsv
2017-01-14 06:47:48 4.79MB Standard oss://oss-odps-test/tsv_output_folder/.odps/20170113224724
561g9m6csz7/M1_2_0-0.tsv
...
```

The *tsv_output_folder* folder in OSS bucket *oss-odps-test* specified by LOCATION contains a *.odps* folder that includes some *.tsv* files and a *.meta* file. Similar file structures are specific to export of data from MaxCompute to OSS:

- When you use MaxCompute to execute INSERT INTO/OVERWRITE on an external table and write data to an OSS storage path, all data is written to a *.odps* folder in the specified LOCATION.
- The *.meta* file in the *.odps* folder is an extra macro data file written by MaxCompute to record valid data in the current folder. If the INSERT operation is successful, all data in the current folder is valid. You are only required to parse the macro data if a job fails. If a job fails or is terminated, you can re-execute the INSERT OVERWRITE statement.
- If the external table is a partitioned table, a corresponding partition subdirectory will be generated under the *tsv_output_folder* folder based on the partition value specified by the INSERT statement. The partition subdirectory contains the *.odps* folder. Example: `test/tsv_output_folder/first-level partition name = partition value/n-level partition name = partition value/.odps/20170113224*****/M1_2_0-0.tsv`

The number of files generated by a built-in StorageHandler in MaxCompute is the same as the number of concurrent jobs in the corresponding SQL stage.

If the `INSERT OVERWRITE ... SELECT ... FROM ... ;` operation allocates 1,000 mappers on source data table *from_tablename*, 1,000 TSV or CSV files will be generated.

Export data to OSS through a custom StorageHandler

In addition to built-in StorageHandlers used to export TSV and CSV files to OSS, MaxCompute also provides an SDK using an unstructured data processing framework to export data files in custom formats.

To export data to OSS, you must first create an external table and then perform an INSERT operation on the external table in a similar manner as using a built-in StorageHandler. However, you must specify a custom StorageHandler in the STORED BY clause when creating the external table.

Note The MaxCompute unstructured data processing framework uses the StorageHandler API to describe the processing of various data storage formats. Specifically, StorageHandler acts as a wrapper class that allows you to specify a custom Extractor and Outputter. An Extractor reads, parses, and extracts data, while an Outputter processes and exports data. A custom StorageHandler must inherit OdpsStorageHandler to implement the getExtractorClass and getOutputterClass APIs.

Using the TextStorageHandler example that customizes an Extractor to access OSS as an example, the following section demonstrates how MaxCompute can export data to OSS text files through a custom StorageHandler, and use | as the column delimiter and \n as the line break. For more information, see [Access OSS unstructured data](#).

Note After [MaxCompute Studio](#) is configured with [MaxCompute Java Module](#), you can view the corresponding sample code in examples. Alternatively, click [aliyun-odps-java-sdk](#) to view the complete code.

- Define an Outputter

The output logic must implement an Outputter API.

```
package com.aliyun.odps.examples.unstructured.text;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.io.OutputStreamSet;
import com.aliyun.odps.io.SinkOutputStream;
import com.aliyun.odps.udf.DataAttributes;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.Outputter;
import java.io.IOException;
public class TextOutputter extends Outputter {
    private SinkOutputStream outputStream;
    private DataAttributes attributes;
    private String delimiter;
    public TextOutputter () {
        // default delimiter, this can be overwritten if a delimiter is provided through the attributes.
        this.delimiter = "|";
    }
    @Override
    public void output(Record record) throws IOException {
        this.outputStream.write(recordToString(record).getBytes());
    }
    // no particular usage of execution context in this example
```

```

@Override
public void setup(ExecutionContext ctx, OutputStreamSet outputStreamSet, DataAttributes attributes) throws IOException {
    this.outputStream = outputStreamSet.next();
    this.attributes = attributes;
    this.delimiter = this.attributes.getValueByKey("delimiter");
    if ( this.delimiter == null)
    {
        this.delimiter=",";
    }
    System.out.println("Extractor using delimiter [" + this.delimiter + "].") ;
}
@Override
public void close() {
    // no-op
}
private String recordToString(Record record){
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < record.getColumnCount(); i++)
    {
        if (null == record.get(i)){
            sb.append("NULL");
        }
        else{
            sb.append(record.get(i).toString());
        }
        if (i != record.getColumnCount() - 1){
            sb.append(this.delimiter);
        }
    }
    sb.append("\n");
    return sb.toString();
}
}

```

There are three Outputter APIs: setup, output, and close, which correspond to three Extractor APIs: setup, extract, and close. setup () and close () are called only once in an Outputter. You can perform initialization in the setup API. The three parameters returned by setup() must be saved as the Outputter class variable to be used in the output() or close() API. The close () API is used to mark the end of the code.

Data processing occurs in the `output(Record)` API. The MaxCompute system calls `output(Record)` once based on each input record processed by the current Outputter. Assume that when an `output(Record)` call returns, the code has already consumed the Record. After the current `output(Record)` call returns, the system will use the memory used by the Record for other purposes. Therefore, when the information in Record is used across multiple `output()` function calls, you must call the `record.clone()` method to save the current record.

 **Note** When you use a custom `StorageHandler` for an external table to implement an Outputter API, the record passed into `Outputter.output(Record record)` is the record generated by the previous operator of the Outputter. In this case, the column name has changed. Column names are not fixed. For example, the column name generated by the `some_function(column_a)` expression is a temporary column name.

Therefore, we recommend that you use `record.get (index)` instead of `record.get (column name)` to obtain the content of a column.

- Define an Extractor

An Extractor is used to read, parse, and process data. If the output tables do not need to be read through MaxCompute, you do not need to define an Extractor.

```
package com.aliyun.odps.examples.unstructured.text;
import com.aliyun.odps.Column;
import com.aliyun.odps.data.ArrayRecord;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.io.InputStreamSet;
import com.aliyun.odps.udf.DataAttributes;
import com.aliyun.odps.udf.ExecutionContext;
import com.aliyun.odps.udf.Extractor;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
/**
 * Text extractor that extract schematized records from formatted plain-text(csv, tsv etc.)
 **/
public class TextExtractor extends Extractor {
    private InputStreamSet inputs;
    private String columnDelimiter;
    private DataAttributes attributes;
    private BufferedReader currentReader;
    private boolean firstRead = true;
    public TextExtractor() {
        // default to ",", this can be overwritten if a specific delimiter is provided (via DataAttributes)
        this.columnDelimiter = ",";
    }
}
```

```
,
// no particular usage for execution context in this example
@Override
public void setup(ExecutionContext ctx, InputStreamSet inputs, DataAttributes attributes) {
    this.inputs = inputs;
    this.attributes = attributes;
    // check if "delimiter" attribute is supplied via SQL query
    String columnDelimiter = this.attributes.getValueByKey("delimiter");
    if (columnDelimiter != null)
    {
        this.columnDelimiter = columnDelimiter;
    }
    System.out.println("TextExtractor using delimiter [" + this.columnDelimiter + "].");
    // note: more properties can be initied from attributes if needed
}
@Override
public Record extract() throws IOException {
    String line = readNextLine();
    if (line == null) {
        return null;
    }
    return textLineToRecord(line);
}
@Override
public void close(){
    // no-op
}
private Record textLineToRecord(String line) throws IllegalArgumentException
{
    Column[] outputColumns = this.attributes.getRecordColumns();
    ArrayRecord record = new ArrayRecord(outputColumns);
    if (this.attributes.getRecordColumns().length != 0){
        // string copies are needed, not the most efficient one, but suffice as an example here
        String[] parts = line.split(columnDelimiter);
        int[] outputIndexes = this.attributes.getNeededIndexes();
        if (outputIndexes == null){
            throw new IllegalArgumentException("No outputIndexes supplied.");
        }
        if (outputIndexes.length != outputColumns.length){
            throw new IllegalArgumentException("Mismatched output schema: Expecting "
                + outputColumns.length + " columns but get " + parts.length);
        }
    }
}
```

```

    }
    int index = 0;
    for(int i = 0; i < parts.length; i++){
        // only parse data in columns indexed by output indexes
        if (index < outputIndexes.length && i == outputIndexes[index]){
            switch (outputColumns[index].getType()) {
                case STRING:
                    record.setString(index, parts[i]);
                    break;
                case BIGINT:
                    record.setBigint(index, Long.parseLong(parts[i]));
                    break;
                case BOOLEAN:
                    record.setBoolean(index, Boolean.parseBoolean(parts[i]));
                    break;
                case DOUBLE:
                    record.setDouble(index, Double.parseDouble(parts[i]));
                    break;
                case DATETIME:
                case DECIMAL:
                case ARRAY:
                case MAP:
                default:
                    throw new IllegalArgumentException("Type " + outputColumns[index].getType() + " not supported for now.");
            }
            index++;
        }
    }
    return record;
}

/**
 * Read next line from underlying input streams.
 * @return The next line as String object. If all of the contents of input
 * streams has been read, return null.
 */
private String readNextLine() throws IOException {
    if (firstRead) {
        firstRead = false;
        // the first read, initialize things

```

```
        currentReader = moveToNextStream();
        if (currentReader == null) {
            // empty input stream set
            return null;
        }
    }
    while (currentReader != null) {
        String line = currentReader.readLine();
        if (line != null) {
            return line;
        }
        currentReader = moveToNextStream();
    }
    return null;
}

private BufferedReader moveToNextStream() throws IOException {
    InputStream stream = inputs.next();
    if (stream == null) {
        return null;
    } else {
        return new BufferedReader(new InputStreamReader(stream));
    }
}
}
```

For more information, see [Access OSS unstructured data](#).

- Define a StorageHandler


```
package com.aliyun.odps.examples.unstructured.text;
import com.aliyun.odps.udf.Extractor;
import com.aliyun.odps.udf.OdpsStorageHandler;
import com.aliyun.odps.udf.Outputter;
public class TextStorageHandler extends OdpsStorageHandler {
    @Override
    public Class<? extends Extractor> getExtractorClass() {
        return TextExtractor.class;
    }
    @Override
    public Class<? extends Outputter> getOutputterClass() {
        return TextOutputter.class;
    }
}
```

If the table does not need to be read, you do not need to specify an Extractor API.

- Compile the code into a package

Compile your custom code into a JAR package and upload it to MaxCompute. If the JAR package is named *odps-TextStorageHandler.jar*, the code used to upload the package as a MaxCompute resource is as follows:

```
add jar odps-TextStorageHandler.jar;
```

- Create an external table

To export data to OSS, you must first create an external table in a similar manner as using a built-in StorageHandler. However, you must specify a custom StorageHandler to export data to the external table.

```
CREATE EXTERNAL TABLE IF NOT EXISTS output_data_txt_external
(
  vehicleId int,
  recordId int,
  patientId int,
  calls int,
  locationLatitude double,
  locationLongitude double,
  recordTime string,
  direction string
)
STORED BY 'com.aliyun.odps.examples.unstructured.text.TextStorageHandler'
WITH SERDEPROPERTIES(
  'delimiter'='|'
  [, 'odps.properties.rolearn'='${roleran}'])
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/'
USING 'odps-TextStorageHandler.jar';
```

 **Note** If the `odps.properties.rolearn` property is required, see [Access OSS unstructured data](#) for more information about *custom authorization* based on STS. If the property is not required, refer to *one-click authorization* or use a plaintext AccessKey pair on LOCATION.

- Perform an INSERT operation on an external table to export data to OSS

After you create an external table and associate it with an OSS storage path through a custom StorageHandler, you can perform standard SQL operations such as INSERT OVERWRITE/INTO on the external table to export data to OSS in the same way as a built-in StorageHandler.

```
INSERT OVERWRITE|INTO TABLE <external_tablename> [PARTITION (partcol1=val1, partcol2=val2 ...)]
select_statement
FROM <from_tablename>
[WHERE where_condition];
```

After the INSERT operation is complete, a series of files will be generated in the `.odps` folder in the OSS storage path specified by LOCATION. This is similar to what happens when you use a built-in StorageHandler.

5.8. Access Table Store (OTS) data

This topic describes how to import data from Table Store (OTS) to the MaxCompute computing environment to achieve seamless connections between multiple data sources.

You can use DataWorks to visually create, search, query, configure, process, and analyze external tables in the MaxCompute environment. For more information, see [External tables](#).

Table Store is a NoSQL database service that built on Alibaba Cloud's Apsara distributed file system, enabling you to store and access massive volumes of structured data in real time. For more information about Table Store, see [What is Table Store](#).

MaxCompute and Table Store are two independent big data computing and storage services. Therefore, these two services must ensure that the network between them is open. When MaxCompute's public cloud service accesses data stored in Table Store, we recommend that you use Table Store's *private network* address, usually a host name suffixed `ots-internal.aliyuncs.com`, for example, `tablestore://odps-ots-dev.cn-shanghai.ots-internal.aliyuncs.com`.

Both Table Store and MaxCompute have their own type systems. The following table describes the data type associations when you process Table Store data in MaxCompute.

MaxCompute Type	Table Store Type
STRING	STRING
BIGINT	INTEGER
DOUBLE	Double
BOOLEAN	BOOLEAN
BINARY	BINARY

Authorization with STS mode

To access Table Store data, MaxCompute requires a secure authorization channel. Therefore, MaxCompute integrates Alibaba Cloud Resource Access Management (RAM) and Token Service (STS) to implement secure data access.

You can authorize permissions in the following two ways:

- When the MaxCompute and Table Store's owner are the same account, you can directly log on with the Alibaba Cloud account and [click here to complete authorization](#).
- Custom authorization
 - i. Firstly, you must grant Table Store access permission to MaxCompute in the RAM console.
Log on to the [RAM console](#) (if MaxCompute and Table Store are not the same account, you must log on with the Table Store account to authorize), and create the role `AliyunODPSDefaultRole` or `AliyunODPSRoleForOtherUser`.
 - ii. Set its policy content as follows:

--If MaxCompute and Table Store use the same account, do the following:

```
{
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}
```

--If MaxCompute and Table Store use different accounts, do the following:

```
{
  "Statement ":[
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "MaxCompute's Owner cloud account UID@odps.aliyuncs.com"
        ]
      }
    }
  ],
  "Version": "1"
}
```

 **Note** On the upper-right corner, click the avatar to open the **Billing Management** page, and then check the account UID.

□
□

iii. Edit this role's authorization policy AliyunODPSRolePolicy:

```
{
  "Version": "1",
  "Statement ":[
  {
    "Action ":[
      "ots:ListTable",
      "ots:DescribeTable",
      "ots:GetRow",
      "ots:PutRow",
      "ots:UpdateRow",
      "ots>DeleteRow",
      "ots:GetRange",
      "ots:BatchGetRow",
      "ots:BatchWriteRow",
      "ots:ComputeSplitPointsBySize"
    ],
    "Resource": "*",
    "Effect": "Allow"
  }
]
}
--You can also customize other permissions
```

iv. Grant the permission AliyunODPSRolePolicy to this role.

Create an external table

In MaxCompute, after creating an external table and introducing the Table Store table data descriptions to the MaxCompute meta system, you can process Table Store data. The following example demonstrates the concept and practice that used in MaxCompute's Table Store access.

Use following statements to create an external table:

```
DROP TABLE IF EXISTS ots_table_external;
CREATE EXTERNAL TABLE IF NOT EXISTS ots_table_external
(
  odps_orderkey bigint,
  odps_orderdate string,
  odps_custkey bigint,
  odps_orderstatus string,
  odps_totalprice double
)
STORED BY 'com.aliyun.odps.TableStoreStorageHandler' --(1)
WITH SERDEPROPERTIES ( --(2)
  'tablestore.columns.mapping'=':o_orderkey,:o_orderdate,o_custkey,o_orderstatus,o_totalprice', -- (a)
  'tablestore.table.name'='ots_tpch_orders' --(b)
  'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole' --(c)
)
LOCATION 'tablestore://odps-ots-dev.cn-shanghai.ots-internal.aliyuncs.com'; --(3)
```

The statement is described as follows:

- `com.aliyun.odps.TableStoreStorageHandler` is MaxCompute's built-in `StorageHandler` for processing Table Store data. It defines the interaction between MaxCompute and Table Store. The relevant logic is implemented by MaxCompute.
- `SERDEPROPERTIES` is an interface that provides parameter options. When using `TableStoreStorageHandler`, three options must be specified, *tablestore.columns.mapping*, *tablestore.table.name*, and *odps.properties.rolearn*.
 - i. *tablestore.columns.mapping option*: describes the columns of the table store table that MaxCompute is going to access, including primary key and attribute columns.
 - At the beginning of the column name, `:` indicates a Table Store primary key. In this example, `:o_orderkey` and `:o_orderdate` are primary key columns and all others are attribute columns.
 - Table Store supports up to 4 primary keys. Primary keys support the string, integer, and binary data types. The first primary key is the partition key.
 - When specifying a mapping relationship, you must provide all the primary keys of the specified Table Store table, but you do not have to provide all attribute columns, only the attribute columns you must access by using MaxCompute. The attribute columns you provide must belong to the Table Store table. Otherwise, an error occurs when you query the created external table.
 - ii. *tablestore.table.name*: the name of the table store table that needs to be accessed. If you specify an incorrect Table Store table name (such as a table that does not exist), the system reports an error. MaxCompute does not create a new Table Store table with the specified name.
 - iii. *odps.properties.rolearn*: ARN information in RAM's `AliyunODPSDefaultRole`. You can get it through **RAM Roles** in the RAM console.

- **LOCATION Clause:** specific information for specifying Table Store information such as instance names and endpoints. Secure access to Table Store data is based on RAM/STS authorization.

If you want to view the created external table structure, run the following statement:

```
desc extended <table_name>;
```

In the returned information, `Extended Info` contains external table information such as `StorageHandler` and `Location`.

Access table data by using an external table

After creating an external table, you can introduce Table Store data to the MaxCompute environment, in which you can use MaxCompute SQL syntax to access Table Store data as follows:

```
SELECT odps_orderkey, odps_orderdate, SUM(odps_totalprice) AS sum_total
FROM ots_table_external
WHERE odps_orderkey > 5000 AND odps_orderkey < 7000 AND odps_orderdate >= '1996-05-03' AND odps_orderdate < '1997-05-01'
GROUP BY odps_orderkey, odps_orderdate
HAVING sum_total > 400000.0;
```

When using the MaxCompute SQL syntax, all of the accessed Table Store details are processed in MaxCompute. This includes column name selection. For example, the column names used in the preceding SQL statements (such as `odps_orderkey` and `odps_totalprice`) are not the original primary key names (`o_orderkey`) or attribute column names (`o_totalprice`) used in Table Store. This is because mapping was already created in the DDL statement used to create the external table. You can retain the original Table Store primary key and column names when creating the external table.

If you perform *multiple computations* on a single data set, instead of remotely reading data from Table Store each time, you can import all the necessary data to MaxCompute, to create a MaxCompute (internal) table. For example:

```
CREATE TABLE internal_orders AS
SELECT odps_orderkey, odps_orderdate, odps_custkey, odps_totalprice
FROM ots_table_external
Where fig > 5000;
```

Currently, `internal_orders` is a MaxCompute table, with all features of a MaxCompute internal table, including an efficiently compressed column storage data format and complete internal macro data, and statistics information. Furthermore, because the data is stored in MaxCompute, the access speed is faster than when accessing external Table Store data. This is especially suitable for hotspot data that is frequently computed.

Export MaxCompute data to Table Store

 **Note** MaxCompute does not directly create external Table Store tables. Therefore, before outputting data to a Table Store table, you must make sure this table has been created (otherwise, the system reports an error).

In the preceding operations, the external table `ots_table_external` has been created to connect MaxCompute with the Table Store table `ots_tpch_orders`, and data has been stored in the internal MaxCompute table `internal_orders`. If you need to write the processed data from `internal_orders` back to Table Store, run *INSERT OVERWRITE TABLE* on the external table as follows:

```
INSERT OVERWRITE TABLE ots_table_external
SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custkey, 'SHIPPED'), CEIL(odps_totalprice)
FROM internal_orders;
```

 **Note** If the data in the ODPS table has a certain order, such as data sorting according to the primary key, a certain OTS partition may be overburdened when you write data to the OTS table. In this case, the advantage of the distributed writes feature cannot be fully used. We recommend that you distribute data to different partitions through *distribute by rand()* before you write data to the OTS table.

```
INSERT OVERWRITE TABLE ots_table_external
SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custkey, 'SHIPPED'), CEIL(odps_totalprice)
FROM (SELECT * FROM internal_orders DISTRIBUTE BY rand()) t;
```

Because Table Store is a NoSQL storage medium of KV data, the data output from MaxCompute only affects the rows with the corresponding primary keys. In this example, the output only affects data in rows with corresponding `dps_orderkey` and `odps_orderdate` primary key values. In addition, in Table Store rows, only the attribute columns specified during external table (`ots_table_external`) creation are updated. Data columns that do not appear in the external table are not modified.

 Note

- MaxCompute data you want to write to OTS cannot exceed 4 MB for one data writing. If the data exceeds 4 MB, you must remove oversized data and write data again, which may result in an error:

```
ODPS-0010000:System internal error - Output to TableStore failed with exception:
TableStore BatchWrite request id XXXXX failed with error code OTSPParameterInvalid and m
essage:The total data size of BatchWriteRow request exceeds the limit
```

- Writing data in bulk or by branch is counted as one data write operation. For more information, see [BatchWriteRow](#). If the volume of bulk write data is too large, you can also branch write.
- When writing data in bulk, do not write duplicate rows. Otherwise, the following error may be reported:

```
Errorcode: FIG, errormessage: the input parameter is invalid
```

For more information, see [Error OTSPParameterInvalid returned when BatchWriteRow is used to commit 100 data records at a time](#).

6. PyODPS

6.1. Quick start

PyODPS is MaxCompute SDK for Python. It provides easy-to-use Python programming interfaces. Similar to pandas, PyODPS provides fast, flexible, and expressive data structures. You can use the data processing feature of PyODPS, which is similar to that of pandas, by calling the DataFrame API provided by PyODPS. This topic describes how to use PyODPS in your projects.

Prerequisites

- MaxCompute is activated. For more information, see [Activate MaxCompute](#).
- DataWorks is activated and a workspace is created. For more information, see [Create a project](#).

Context

You can develop most programs in MaxCompute by using SQL statements. However, you must use Python to develop complex business logic and user defined functions (UDFs). For example, you must use Python in the following scenarios:

- Interface connection

An external service must provide required information to complete authentication before the service can access any data record in MaxCompute tables by using an HTTP interface.
- Asynchronous calls

Generally, the system creates a node for each of thousands of tasks with similar data processing logic. These nodes are difficult to manage and occupy excessive resources at the same time. PyODPS allows you to use queues to asynchronously run SQL tasks at a high concurrency. Queues also help you manage all nodes in a unified manner.
- UDF development

If the built-in functions of MaxCompute cannot meet your requirements, you can develop UDFs. For example, you can develop a UDF to format data of the DECIMAL type with thousands separators.
- SQL performance improvement

To check whether database transactions follow the first in, first out (FIFO) rule, you must compare each new record with historical records in the only transaction table. Generally, this transaction table contains a large number of records. If you use SQL statements to complete the comparison, the time complexity is $O(N^2)$. In addition, the SQL statements may fail to return the expected result. To resolve this issue, you can call the cache method in Python code to traverse records in the transaction table only once. In this case, the time complexity is $O(N)$ and the efficiency is improved significantly.
- Other scenarios

You can use Python to develop an amount allocation model. For example, you can develop a model to allocate USD 10 to three persons. In this model, you must define the logic for handling the cash over and short.

For more information about the application scenarios of PyODPS, see the following topics:

- [Quick start](#)
- [Use a PyODPS node to read data from a partitioned table](#)
- [Use a PyODPS node to pass parameters](#)
- [Reference a third-party package in a PyODPS node](#)
- [Use a PyODPS node to read data from the level-1 partition of the specified table](#)
- [Use a PyODPS node to query data based on specific criteria](#)
- [Use a PyODPS node to perform sequence operations](#)

Procedure

1. Create a PyODPS node. This section describes how to create a PyODPS 2 node in the DataWorks console. For more information, see [PyODPS node](#).

Note

- The Python version of a PyODPS node is 2.7.
- Each PyODPS node can process a maximum of 50 MB data and can occupy a maximum of 1 GB memory. Otherwise, DataWorks terminates the PyODPS node. Do not write Python code that will process an extra large amount of data in PyODPS nodes.
- Writing and debugging code in DataWorks is inefficient. We recommend that you install IntelliJ IDEA locally to write code.

- i. Create a workflow. Log on to the DataWorks console and go to the [Data Development](#) tab. Right-click **Business process** and select **New business process**.
- ii. Create a PyODPS node. Right-click the workflow that you created in the workflow list and choose **New > MaxCompute > PyODPS 2**. In the New node dialog box, set the Node name parameter and click **Submit**.

2. Configure and run the PyODPS 2 node.

- i. Write the code of the PyODPS 2 node.

Write the test code in the code editor of the PyODPS 2 node. In this example, write the following code in the code editor, which covers a full range of table operations. For more information about table operations and SQL operations, see [Tables](#) and [SQL](#).

```
from odps import ODPS
import sys

reload(sys)
# Set UTF-8 as the default encoding format. You must execute this statement if the target data contains Chinese characters.
sys.setdefaultencoding('utf8')

# Create a non-partitioned table named my_new_table, which contains the fields with the specified names and of the specified data types.
table = o.create_table('my_new_table', 'num bigint, id string', if_not_exists=True)

# Write data to the my_new_table table.
records = [[111, 'aaa'],
            [222, 'bbb'],
            [333, 'ccc'],
            [444, 'Chinese']]
o.write_table(table, records)

# Read data from the my_new_table table.
for record in o.read_table(table):
    print record[0], record[1]

# Read data from the my_new_table table by executing an SQL statement.
result = o.execute_sql('select * from my_new_table;', hints={'odps.sql.allow.fullscan': 'true'})

# Obtain the execution result of the SQL statement.
with result.open_reader() as reader:
    for record in reader:
        print record[0], record[1]

# Delete the table.
table.drop()
```

- ii. Run the code. After you write the code, click in the toolbar. After the code is run, you can view the running result of the PyODPS 2 node on the Run Log tab. The result in the following figure indicates that the running is successful.



6.2. Installation guide and limits

PyODPS is MaxCompute SDK for Python. It provides the DataFrame framework and basic operations on MaxCompute objects for you to analyze data in MaxCompute by using Python. This topic describes how to install PyODPS and its limits.

Prerequisites

The following requirements are met before you install PyODPS:

- The setuptools version is 3.0 or later.
- The Requests version is 2.4.0 or later.

Procedure

1. Install pip. We recommend that you use pip, a Python package manager, to install PyODPS. For more information, see [Installation](#). The following installation commands are for your reference:

```
pip install setuptools>=3.0
pip install requests>=2.4.0
pip install greenlet>=0.4.10 # Optional. It accelerates Tunnel-based data upload.
pip install cython>=0.19.0 # Optional. We recommend that you do not install Cython if you use a Windows operating system.
```

Note

- We recommend that you use Alibaba Cloud images to accelerate data download. For more information about Alibaba Cloud images, see [Index of /pypi/](#).
- If you use a Windows operating system, make sure that you have installed Visual C++ and Cython of correct versions. Otherwise, you cannot accelerate Tunnel-based data upload. For more information about the versions of Visual C++ and Cython, see [WindowsCompilers](#).

2. Run the following command to install PyODPS:

```
pip install pyodps
```

3. Run the following command to check whether the installation is successful:

```
python -c "from odps import ODPS"
```

4. If the Python version is not the default, you can run the following command to switch to the default version after you have installed pip:

```
/home/tops/bin/python2.7 -m pip install setuptools>=3.0
```

Limits

- MaxCompute SQL statements have limits. For more information, see [MaxCompute SQL limits](#).
- PyODPS nodes in DataWorks have limits. For more information, see [Use PyODPS in DataWorks](#).
- Restricted by the sandbox, some programs that are debugged locally by using the pandas computing backend cannot be debugged in MaxCompute.

What to do next

We recommend that you install the following tools to accelerate Tunnel-based data upload:

- Greenlet 0.4.10 or later
- Cython 0.19.0 or later

6.3. Platform instructions

6.3.1. Overview

PyODPS can be called as a data development node on a data development platform such as DataWorks. The platform provides a PyODPS running environment and can schedule and run nodes. You do not need to manually create a MaxCompute entry object. To migrate PyODPS nodes from a data development platform to a manually deployed PyODPS environment, read the instructions in the following topics:

- [Migrate PyODPS nodes from a data development platform to a local PyODPS environment](#)
- [Use PyODPS in DataWorks](#)

6.3.2. Migrate PyODPS nodes from a data development platform to a local PyODPS environment

If you need to debug PyODPS locally or the resources on the data development platform where PyODPS is deployed cannot meet your requirements, you can migrate PyODPS nodes from the platform to a local PyODPS environment.

To migrate PyODPS nodes from a data development platform to a local PyODPS environment, perform the following steps:

1. Install PyODPS in a local environment. For more information, see [Installation guide and limits](#).
2. Create a MaxCompute entry object in the local environment.

You can execute the following statement on the data development platform to generate a template of the statement that is required to create a MaxCompute entry object. Then, you can modify the template to obtain the required statement.

```
print("\nfrom odps import ODPS\nno = ODPS(%r, '<access-key>', %r, '<endpoint>')\n" % (o.account.access_id, o.project))
```

Parameter description:

- `<access-key>`: the AccessKey pair of your Alibaba Cloud account.
- `<endpoint>`: the endpoint of MaxCompute. For more information, see [Configure endpoints](#).
- **Project name**: You do not need to specify the project name in the statement for creating a MaxCompute entry object if the statement is generated by using the preceding statement.

When you migrate PyODPS nodes from DataWorks to a local environment, you can manually specify a project name in the statement for creating a MaxCompute entry object. Make sure that you understand the difference between workspaces in DataWorks and projects in MaxCompute. A DataWorks workspace in basic mode maps a MaxCompute project named in the <Project name> format, and provides a production environment for jobs in the MaxCompute project. A DataWorks workspace in standard mode maps a MaxCompute project named in the <Project name>_dev format and a MaxCompute project named in the <Project name> format. This workspace provides a development environment for jobs in the MaxCompute project named in the <Project name>_dev format and a production environment for jobs in the MaxCompute project named in the <Project name> format. For more information, see [Basic mode and standard mode](#).

3. Place the obtained statement at the beginning of all code.

6.3.3. Use PyODPS in DataWorks

This topic describes how to use PyODPS in DataWorks and the limits.

For a usage example, see [Use a PyODPS node to segment Chinese text based on Jieba](#).

Create a PyODPS node

You can create a PyODPS node for a workflow.

MaxCompute entry

Each PyODPS node in DataWorks contains the global variable `odps` or `o`, which is the MaxCompute entry. You do not need to specify the MaxCompute entry.

```
print(o.exist_table('pyodps_iris'))
```

Execute SQL statements

You can execute SQL statements in the PyODPS node. For more information, see [SQL](#).

By default, InstanceTunnel is disabled in DataWorks. In this case, `instance.open_reader` is executed by using the Result interface, and a maximum of 10,000 data records can be read. You can use `reader.count` to obtain the number of data records. If you need to obtain all data iteratively, you must disable the limit on the data volume. You can execute the following statements to enable InstanceTunnel and disable the limit.

```
options.tunnel.use_instance_tunnel = True
options.tunnel.limit_instance_tunnel = False # Disable the limit on the data volume.

with instance.open_reader() as reader:
    # Use InstanceTunnel to read all data.
```

You can also add `tunnel=True` to `open_reader` to enable InstanceTunnel for the current `open_reader` operation. You can add `limit=False` to `open_reader` to disable the limit on the data volume for the current `open_reader` operation.

```
with instance.open_reader(tunnel=True, limit=False) as reader:
    # The current open_reader operation is executed by using InstanceTunnel and all data can be read.
```

DataFrame

- Perform operations on DataFrames

To perform operations on **DataFrames** in DataWorks, you must explicitly call automatically executed methods, such as `execute` and `head`. For more information, see [Execution](#).

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
for record in iris[iris.sepal_width < 3].execute(): # Call an automatically executed method to process each data record.
```

To call an automatically executed method for data display, set `options.interactive` to `True`.

```
from odps import options
from odps.df import DataFrame
options.interactive = True # Set options.interactive to True at the beginning of the code.
iris = DataFrame(o.get_table('pyodps_iris'))
print(iris.sepal_width.sum()) # The method is executed immediately when the system displays information.
```

- Display details

To display details, you must set `options.verbose` to `True`. By default, this parameter is set to `True` in DataWorks. The system displays details such as the Logview URL during the running process.

Obtain scheduling parameters

Different from SQL nodes in DataWorks, a PyODPS node does not replace strings such as `${param_name}` in the code. Instead, it adds a dictionary named `args` as a global variable before it runs the code. You can obtain the scheduling parameters from the dictionary. This way, the Python code is not affected. For example, on the **Scheduling configuration** tab of a PyODPS node in DataWorks, you can specify `ds=${yyyymmdd}` in the **Parameters** field in the **Basic properties** section. Then, you can specify the following commands in the code of the node to obtain the parameter value:

```
print('ds=' + args['ds'])
ds=20161116
```


 **Note** You can run the following command to obtain the partition named

```
ds=${yyyymmdd} :
```

```
o.get_table('table_name').get_partition('ds=' + args['ds'])
```

Limits on functions

- The Python version of a PyODPS node is 2.7.
- Each PyODPS node supports processing a maximum of 50 MB data and can occupy a maximum of 1 GB memory. Otherwise, DataWorks terminates the PyODPS node. Do not write unnecessary Python data processing code in PyODPS nodes.
- Writing and debugging code in DataWorks is inefficient. We recommend that you install an IDE locally to write code.
- To avoid excess pressure on the gateway of DataWorks, DataWorks limits the CPU utilization and memory usage. If the system displays **Got killed**, the memory usage exceeds the limit and the system terminates the related processes. Therefore, we recommend that you do not perform local data operations. However, the limits on the memory usage and CPU utilization do not apply to SQL or DataFrame nodes, except to `pandas`, that are initiated by PyODPS.
- Functions may be limited in the following aspects due to the lack of packages such as `matplotlib`:
 - The use of the `plot` function of `DataFrame` is affected.
 - `DataFrame` user defined functions (UDFs) can be executed only after they are submitted to MaxCompute. As required by the Python sandbox, you can only use pure Python libraries and the NumPy library to execute UDFs. Other third-party libraries such as `pandas` cannot be used.
 - However, you can use the NumPy and `pandas` libraries that are pre-installed in DataWorks to execute non-UDFs. You are not allowed to use other third-party libraries that contain binary code.
- For compatibility reasons, `options.tunnel.use_instance_tunnel` is set to `False` in DataWorks by default. If you want to enable `InstanceTunnel` globally, you must set this parameter to `True`.
- For implementation reasons, the Python `atexit` package is not supported. You must use `try-finally` to implement related features.

6.3.4. Create a PyODPS 3 node

This topic describes the usage limits of PyODPS 3 nodes and how to create a PyODPS 3 node in the DataWorks console.

Limits

- Python 3 defines bytecode differently in its different subversions such as Python 3.7 and Python 3.8.

MaxCompute is compatible with Python 3.7. A MaxCompute client that uses another subversion of Python 3 will return an error when it executes code with specific syntax. For example, a MaxCompute client that uses Python 3.8 will return an error when it executes code with the finally block syntax. We recommend that you use a MaxCompute client with Python 3.7.

- Each PyODPS 3 node can process a maximum of 50 MB data and can occupy a maximum of 1 GB memory. Otherwise, DataWorks terminates the PyODPS 3 node. Do not write Python code that will process an extra large amount of data in a PyODPS 3 node.

Create a PyODPS 3 node

1. Go to the DataStudio page.
 - i. Log on to the [DataWorks console](#).
 - ii. In the left-side navigation pane, click **Workspaces**.
 - iii. In the top navigation bar, select the region where the target workspace resides. Find the target workspace and click **Data Analytics** in the Actions column.
2. On the Data Development tab, move the pointer over  and choose **MaxCompute > PyODPS 3**. You can also click the target workflow, right-click **MaxCompute**, and then choose **New > PyODPS 3**.
3. In the New node dialog box, set the **Node name** and **Destination folder** parameters.

 **Note** The node name must be 1 to 128 characters in length and can contain letters, digits, underscores (_), and periods (.). It is not case-sensitive.

4. Click **Submit**.
5. Enter the code of the PyODPS 3 node. Enter the code of the PyODPS 3 node based on your needs. For example, if you need to use the `execute_sql()` method to execute SQL statements in the node, you must specify runtime parameters for the SQL statements.

```
hints={'odps.sql.python.version': 'cp37', 'odps.isolation.session.enable': True}
```

If you need to use a user defined function (UDF), such as `df.map`, `df.map_reduce`, `df.apply`, or `df.agg`, to call the DataFrame API in the node, specify the following setting:

```
hints={'odps.isolation.session.enable': True}
```

PyODPS determines the runtime environment of the UDF and commits SQL statements based on the Python version that the MaxCompute client uses. Assume that the MaxCompute client uses Python 3. If the UDF executes a print statement specific to Python 2 to call the DataFrame API, the MaxCompute client returns the `ScriptError` error.

6. On the configuration tab of the PyODPS 3 node, click the **Scheduling configuration** tab in the right-side navigation pane. On the Scheduling configuration tab, set the scheduling properties of the node. For more information, see [Basic properties](#).
7. Save and commit the node.

 **Notice** You must set the **Rerun attribute** and **Dependent upstream node** parameters on the Scheduling configuration tab before you can commit the node.

- i. Click  in the toolbar to save the node.
- ii. Click  in the toolbar to commit the node.
- iii. In the **Submit New Version** dialog box, enter your comments in the **Change description** field.
- iv. Click **OK**.

In a workspace in standard mode, you must click **Publish** in the upper-right corner after you commit the AnalyticDB for MySQL node. For more information, see [Deploy a node](#).

8. Test the node. For more information, see [Auto triggered nodes](#).

6.4. Basic operations

6.4.1. Overview

PyODPS supports basic operations on MaxCompute objects. You can use Python-compliant programming methods to perform operations on MaxCompute.

PyODPS allows you to perform basic operations on the following MaxCompute objects:

- [Projects](#)
- [Tables](#)
- [SQL](#)
- [Task instances](#)
- [Resources](#)
- [Functions](#)

6.4.2. Projects

This topic describes how to perform basic operations on projects by using PyODPS.

`project.get_project()`

A project is the basic organizational unit of MaxCompute. For more information, see [Project](#).

You can perform the following basic operations on a project:

- **Obtain a project:** You can call the `get_project()` method of a MaxCompute entry object to obtain a specific project.

```
project = o.get_project('my_project') # Obtain the specified project.  
project = o.get_project()             # Obtain the current project.
```

To obtain a specific project, you must set the `project_name` parameter to the name of the project for the `get_project()` method. If you do not specify a project name for the method, the method returns the current project.

- **Check whether a project exists:** You can call the `exist_project()` method to check whether a specific project exists.

6.4.3. Tables

This topic describes the basic operations as well as other operations that you can perform on a MaxCompute table. For example, you can create a table, create a table schema, synchronize table updates, obtain table data, delete a table, manage table partitions, and convert a table to a DataFrame.

Basic operations

A table is the data storage unit in MaxCompute. For more information, see [表](#). You can perform the following basic operations on tables:

- Obtain all tables in the current project by calling the `list_tables()` method of a MaxCompute entry object.

```
# Obtain all tables in the current project.  
for table in o.list_tables():
```

- Check whether a specific table exists by calling the `exist_table()` method of a MaxCompute entry object.
- Obtain a specific table by calling the `get_table()` method of a MaxCompute entry object.

```
t = o.get_table('dual')
t.schema
odps.Schema {
  c_int_a      bigint
  c_int_b      bigint
  c_double_a   double
  c_double_b   double
  c_string_a   string
  c_string_b   string
  c_bool_a     boolean
  c_bool_b     boolean
  c_datetime_a datetime
  c_datetime_b datetime
}
t.lifecycle
-1
print(t.creation_time)
2014-05-15 14:58:43
t.is_virtual_view
False
t.size
1408
t.comment
'Dual Table Comment'
t.schema.columns
[<column c_int_a, type bigint>,
 <column c_int_b, type bigint>,
 <column c_double_a, type double>,
 <column c_double_b, type double>,
 <column c_string_a, type string>,
 <column c_string_b, type string>,
 <column c_bool_a, type boolean>,
 <column c_bool_b, type boolean>,
 <column c_datetime_a, type datetime>,
 <column c_datetime_b, type datetime>]
t.schema['c_int_a']
<column c_int_a, type bigint>
t.schema['c_int_a'].comment
'Comment of column c_int_a'
```

You can obtain tables across projects by specifying the `project` parameter.

```
t = o.get_table('dual', project='other_project')
```

Create a table schema

You can use one of the following methods to create a table schema:

- Create a schema based on table columns and optional partitions.

```
from odps.models import Schema, Column, Partition
columns = [Column(name='num', type='bigint', comment='the column'),
           Column(name='num2', type='double', comment='the column2')]
partitions = [Partition(name='pt', type='string', comment='the partition')]
schema = Schema(columns=columns, partitions=partitions)
schema.columns
[<column num, type bigint>,
 <column num2, type double>,
 <partition pt, type string>]
schema.partitions
[<partition pt, type string>]
schema.names # Obtain the names of non-partitioned fields.
['num', 'num2']
schema.types # Obtain the data types of non-partitioned fields.
[bigint, double]
```

- Create a schema by calling the `Schema.from_lists()` method. This method is more convenient, but you cannot directly set comments for columns and partitions.

```
schema = Schema.from_lists(['num', 'num2'], ['bigint', 'double'], ['pt'], ['string'])
schema.columns
[<column num, type bigint>,
 <column num2, type double>,
 <partition pt, type string>]
```

Create a table

You can call the `create_table()` method to create a table by using one of the following methods:

- Use a table schema to create a table.

```
table = o.create_table('my_new_table', schema)
table = o.create_table('my_new_table', schema, if_not_exists=True) # Create the table only if no table with the same name exists.
table = o.create_table('my_new_table', schema, lifecycle=7) # Set the lifecycle of the table.
```

- Create a table by specifying the names and data types of the fields to be contained in the table.

```
# Create a non-partitioned table.
table = o.create_table('my_new_table', 'num bigint, num2 double', if_not_exists=True)

# Create a partitioned table with the specified table columns and partition fields.
table = o.create_table('my_new_table', ('num bigint, num2 double', 'pt string'), if_not_exists=True)
```

By default, when you create a table, you can only use the **BIGINT**, **DOUBLE**, **DECIMAL**, **STRING**, **DATETIME**, **BOOLEAN**, **MAP**, and **ARRAY** data types. If you need to use other data types such as **TINYINT** and **STRUCT**, you must set `options.sqluse_odps2_extension` to `True`. Example:

```
from odps import options
options.sqluse_odps2_extension = True

table = o.create_table('my_new_table', 'cat smallint, content struct<title:varchar(100), body:string>')
```

Synchronize table updates

After another program updates a table, for example, the table schema, you can call the `reload()` method to synchronize the update.

```
table.reload()
```

Insert a record to a table

A record refers to a single row in a table. You can call the `new_record()` method to insert a record to a specific table.

```
t = o.get_table('mytable')
r = t.new_record(['val0', 'val1']) # The number of values must be equal to the number of fields in the table schema.
r2 = t.new_record() # You can leave the value empty.
r2[0] = 'val0' # Set a value based on an offset.
r2['field1'] = 'val1' # Set a value based on the field name.
r2.field1 = 'val1' # Set a value based on an attribute.

print(record[0]) # Obtain the value at position 0.
print(record['c_double_a']) # Obtain a value based on a field.
print(record.c_double_a) # Obtain a value based on an attribute.
print(record[0: 3]) # Slice data.
print(record[0, 2, 3]) # Obtain values at multiple positions.
print(record['c_int_a', 'c_double_a']) # Obtain values based on multiple fields.
```

Write data to a table

- Call the `write_table()` method of a MaxCompute entry object to write data to a table.

```
records = [[111, 'aaa', True],          # A list can be specified.
            [222, 'bbb', False],
            [333, 'ccc', True],
            [444, 'Chinese', False]]
o.write_table('test_table', records, partition='pt=test', create_partition=True)
```

 Note

- Each time you call the `write_table()` method, MaxCompute generates a file on the server. This operation is time-consuming. If a large number of files are concurrently generated, subsequent queries will be less efficient. We recommend that you write multiple records at a time or provide a generator object if you use the `write_table()` method.
- If you call the `write_table()` method to write data to a table, new data will be appended to existing data. PyODPS does not provide options to overwrite existing data. You must manually delete the data that you want to overwrite. For a non-partitioned table, you must call the `table.truncate()` method to delete data. For a partitioned table, you must delete partitions first and then create partitions again.

- Call the `open_writer()` method to write data to a table.

```
with t.open_writer(partition='pt=test') as writer:
    records = [[111, 'aaa', True],          # A list can be specified.
                [222, 'bbb', False],
                [333, 'ccc', True],
                [444, 'Chinese', False]]
    writer.write(records) # records can be iterable objects.

with t.open_writer(partition='pt1=test1,pt2=test2') as writer: # Write data in multi-level partitioning
    mode.

    records = [t.new_record([111, 'aaa', True]), # Record objects can be used.
                t.new_record([222, 'bbb', False]),
                t.new_record([333, 'ccc', True]),
                t.new_record([444, 'Chinese', False])]
    writer.write(records)
```

If the specified partition does not exist, set the `create_partition` parameter to True to create a partition. Example:


```
with t.open_writer(partition='pt=test', create_partition=True) as writer:
    records = [[111, 'aaa', True],          # A list can be specified.
               [222, 'bbb', False],
               [333, 'ccc', True],
               [444, 'Chinese', False]]
    writer.write(records) # records can be iterable objects.
```

- Use multiple processes to concurrently write data to a table.

If multiple processes concurrently write data to a table, each process shares the same session ID but writes data in a separate block. Each block maps a file on the server. After all the processes finish writing data, the main process submits the data.

```
import random
from multiprocessing import Pool
from odps.tunnel import TableTunnel

def write_records(session_id, block_id):
    # Create a session with the specified session ID.
    local_session = tunnel.create_upload_session(table.name, upload_id=session_id)
    # Create a writer with the specified block ID.
    with local_session.open_record_writer(block_id) as writer:
        for i in range(5):
            # Generate data and write the data to the corresponding block.
            record = table.new_record([random.randint(1, 100), random.random()])
            writer.write(record)

if __name__ == '__main__':
    N_WORKERS = 3

    table = o.create_table('my_new_table', 'num bigint, num2 double', if_not_exists=True)
    tunnel = TableTunnel(o)
    upload_session = tunnel.create_upload_session(table.name)

    # Each process uses the same session ID.
    session_id = upload_session.id

    pool = Pool(processes=N_WORKERS)
    futures = []
    block_ids = []
    for i in range(N_WORKERS):
        futures.append(pool.apply_async(write_records, (session_id, i)))
        block_ids.append(i)
    [f.get() for f in futures]

    # Submit the data in all the blocks.
    upload_session.commit(block_ids)
```

Obtain table data

You can use one of the following methods to obtain data from a table:

- Call the `read_table()` method of a MaxCompute entry object.

```
for record in o.read_table('test_table', partition='pt=test'):
    # Process a record.
```

- Call the `head()` method to obtain less than 10,000 data records from the beginning of a table.

```
t = o.get_table('dual')
# Process each record.
for record in t.head(3):
```

- Call the `open_reader()` method.
 - Open the reader with a `WITH` clause.

```
with t.open_reader(partition='pt=test') as reader:
    count = reader.count
    for record in reader[5:10] # You can execute the statement multiple times until all records are read. The number of records is specified by count. You can change it to parallel-operation code.
        # Process a record.
```

- Open the reader without a `WITH` clause.

```
reader = t.open_reader(partition='pt=test')
count = reader.count
for record in reader[5:10] # You can execute the statement multiple times until all records are read. The number of records is specified by count. You can change it to parallel-operation code.
    # Process a record.
```

Delete a table

You can call the `delete_table()` method to delete an existing table.

```
o.delete_table('my_table_name', if_exists=True) # Delete a table only if the table exists.
t.drop() # Call the drop() method to delete a table if the table exists.
```

Create a DataFrame

PyODPS provides the DataFrame framework, which allows you to conveniently query and manage MaxCompute data. For more information, see [DataFrame](#). You can call the `to_df()` method to convert a table to a DataFrame.

```
table = o.get_table('my_table_name')
df = table.to_df()
```

Manage table partitions

- Check whether a table is partitioned.

```
if table.schema.partitions:
    print('Table %s is partitioned.' % table.name)
```

- Iterate over all the partitions in a table.

```
for partition in table.partitions:
    print(partition.name)
for partition in table.iterate_partitions(spec='pt=test'):
    # Iterate over level-2 partitions.
```

- Check whether a partition exists.

```
table.exist_partition('pt=test,sub=2015')
```

- Obtain a partition.

```
partition = table.get_partition('pt=test')
print(partition.creation_time)
2015-11-18 22:22:27
partition.size
0
```

- Create a partition.

```
t.create_partition('pt=test', if_not_exists=True) # Create a partition only if no partition with the same name exists.
```

- Delete a partition.

```
t.delete_partition('pt=test', if_exists=True) # Delete a partition only if the partition exists.
partition.drop() # Call the drop() method to delete a partition if the partition exists.
```

Upload and download data by using MaxCompute Tunnel

MaxCompute Tunnel is the data tunnel of MaxCompute. You can use Tunnel to upload data to or download data from MaxCompute.

- Example of data upload

```
from odps.tunnel import TableTunnel

table = o.get_table('my_table')

tunnel = TableTunnel(odps)
upload_session = tunnel.create_upload_session(table.name, partition_spec='pt=test')

with upload_session.open_record_writer(0) as writer:
    record = table.new_record()
    record[0] = 'test1'
    record[1] = 'id1'
    writer.write(record)

    record = table.new_record(['test2', 'id2'])
    writer.write(record)

upload_session.commit([0])
```

- Example of data download

```
from odps.tunnel import TableTunnel

tunnel = TableTunnel(odps)
download_session = tunnel.create_download_session('my_table', partition_spec='pt=test')

with download_session.open_record_reader(0, download_session.count) as reader:
    for record in reader:
        # Process each record.
```

 Note

- PyODPS does not allow you to upload data by using foreign tables. For example, you cannot upload data from Object Storage Service (OSS) or Tablestore by using foreign tables.
- We recommend that you use the write and read interfaces of tables instead of Tunnel.
- In a CPython environment, PyODPS compiles C programs during installation to accelerate Tunnel-based upload and download.

6.4.4. SQL

PyODPS supports MaxCompute SQL queries and provides methods to obtain execution results.

Execute SQL statements

You can call the `execute_sql()` and `run_sql()` methods of a MaxCompute entry object to execute SQL statements. The methods return the task instances that execute the SQL statements. For more information, see [Task instances](#).

Parameters

- **statement:** the SQL statement to execute.

 **Note** Not all SQL statements can be executed by using methods of the MaxCompute entry object. Some SQL statements that are executable on the MaxCompute client may fail to be executed by using the `execute_sql()` and `run_sql()` methods. You must use other methods to execute non-DDL or non-DML statements. For example, you must use the `run_security_query` method to execute GRANT or REVOKE statements, and use the `run_xflow` or `execute_xflow` method to run Machine Learning Platform for Artificial Intelligence (PAI) commands.

- **hints:** the runtime parameters. The hints parameter is of the DICT type.

Examples

- Execute SQL statements.

```
o.execute_sql('select * from dual') # Execute the statement in synchronous mode. Other instances are blocked until the SQL statement is executed.
instance = o.run_sql('select * from dual') # Execute the statement in asynchronous mode.
print(instance.get_logview_address()) # Obtain the Logview URL of an instance.
instance.wait_for_success() # Other instances are blocked until the SQL statement is executed.
```

- Set runtime parameters for SQL statements.

```
o.execute_sql('select * from pyodps_iris', hints={'odps.sql.mapper.split.size': 16})
```

If you set the `sql.settings` parameter that is effective globally, the hints parameter is automatically set each time you execute the SQL statement.

```
from odps import options
options.sql.settings = {'odps.sql.mapper.split.size': 16}
o.execute_sql('select * from pyodps_iris') # The hints parameter is automatically set based on your global settings.
```

Obtain the execution results of SQL statements

You can call the `open_reader` method to obtain execution results of SQL statements in the following scenarios:

- The SQL statements return structured data.

```
with o.execute_sql('select * from dual').open_reader() as reader:
    for record in reader:
        # Process each record.
```

- The `DESC` statement is executed. In this case, you can call the `reader.raw` method to obtain the raw execution result.

```
with o.execute_sql('desc dual').open_reader() as reader:
    print(reader.raw)
```

If you specify `options.tunnel.use_instance_tunnel == True` when you call the `open_reader` method, PyODPS calls `InstanceTunnel` by default. However, if you are using an earlier version of MaxCompute or an error occurs when PyODPS calls `InstanceTunnel`, PyODPS generates an alert and automatically downgrades the call object to the old `Result` interface. You can identify the cause of the downgrade based on the alert information. If the result of `InstanceTunnel` does not meet your expectation, set this option to `False`. When you call the `open_reader` method, you can specify whether to call `InstanceTunnel` or the `Result` interface by setting the `tunnel` parameter.

```
# Call InstanceTunnel.
with o.execute_sql('select * from dual').open_reader(tunnel=True) as reader:
    for record in reader:
        # Process each record.
# Call the Result interface.
with o.execute_sql('select * from dual').open_reader(tunnel=False) as reader:
    for record in reader:
        # Process each record.
```

By default, PyODPS does not limit the amount of data that can be read from an instance. However, for a protected project, the amount of data that can be downloaded by using `Tunnel` is limited. If you do not specify `options.tunnel.limit_instance_tunnel`, the limit is automatically enabled, and the number of data records that can be downloaded is limited based on the project configurations. Generally, a maximum of 10,000 data records can be downloaded. You can add a limit option to the `open_reader` method or set `options.tunnel.limit_instance_tunnel` to `True` to manually limit the amount.

If your MaxCompute version only supports the old `Result` interface and you need to read all data, you can export the execution result of the SQL statement to another table and then use the table read interface to read data. This may be limited by project security settings.

In PyODPS V0.7.7.1 and later, you can use the `open_reader` method to call `InstanceTunnel` to obtain all data. For more information, see [InstanceTunnel](#).

```
instance = o.execute_sql('select * from movielens_ratings limit 20000')
with instance.open_reader() as reader:
    print(reader.count)
    # for record in reader: Traverse the 20,000 data records. In this example, only 10 data records are o
    btained based on data slicing.
    for record in reader[:10]:
        print(record)
```

Set an alias

If the resource referenced by a UDF changes dynamically, you can set an alias for the old resource and use the alias as the name of the new resource. This way, you do not need to delete the UDF or create another UDF.


```

from odps.models import Schema
myfunc = ""\
from odps.udf import annotate
from odps.distcache import get_cache_file

@annotate('bigint->bigint')
class Example(object):
    def __init__(self):
        self.n = int(get_cache_file('test_alias_res1').read())

    def evaluate(self, arg):
        return arg + self.n
"""
res1 = o.create_resource('test_alias_res1', 'file', file_obj='1')
o.create_resource('test_alias.py', 'py', file_obj=myfunc)
o.create_function('test_alias_func',
                  class_type='test_alias.Example',
                  resources=['test_alias.py', 'test_alias_res1'])

table = o.create_table(
    'test_table',
    schema=Schema.from_lists(['size'], ['bigint']),
    if_not_exists=True
)

data = [[1, ], ]
# Write a row of data that contains only one value: 1.
o.write_table(table, 0, [table.new_record(it) for it in data])

with o.execute_sql(
    'select test_alias_func(size) from test_table').open_reader() as reader:
    print(reader[0][0])
res2 = o.create_resource('test_alias_res2', 'file', file_obj='2')
# Set the alias of resource res1 as the name of resource res2 without the need to modify the UDF or r
esource.
with o.execute_sql(
    'select test_alias_func(size) from test_table',
    aliases={'test_alias_res1': 'test_alias_res2'}).open_reader() as reader:
    print(reader[0][0])

```

Execute SQL statements in an interactive environment

You can use SQL plug-ins to execute SQL statement or parameterized queries in IPython and Jupyter. For more information, see [IPython](#) and [Jupyter Notebook](#).

Set biz_id

Occasionally, you must specify `biz_id` for the SQL statement to execute. Otherwise, an error occurs. In this case, you can set `biz_id` in the global `options` to troubleshoot the error.

```
from odps import options
options.biz_id = 'my_biz_id'
o.execute_sql('select * from pyodps_iris')
```

6.4.5. Task instances

A task, such as an SQL task, is the basic compute unit in MaxCompute. This topic describes the basic operations on task instances in PyODPS.

Basic operations

When a task is triggered, a MaxCompute instance is created for the task. For more information, see [Instance](#). You can perform the following operations on MaxCompute instances:

- `list_instances()` : obtains all instances in a specific project.
- `exist_instance()` : checks whether a specific instance exists.
- `get_instance()` : obtains a specific instance.
- `stop_instance()` : stops a specific instance.

 **Note** You can also call the `stop()` method to stop a specific instance.

Example

```
for instance in o.list_instances():
    print(instance.id)
o.exist_instance('my_instance_id')
```

Obtain the Logview URL of an instance

- For an SQL task instance, you can call the `get_logview_address` method to obtain its Logview URL.

```
# Execute an SQL statement to obtain a specific instance. Then, call the get_logview_address method to obtain the Logview URL of the instance.
instance = o.run_sql('desc pyodps_iris')
print(instance.get_logview_address())

# Obtain the Logview URL of an instance with the specified instance ID.
instance = o.get_instance('2016042605520945g9k5pvyi2')
print(instance.get_logview_address())
```

- For an XFlow task instance, you must enumerate its sub-instances and then obtain the Logview URL of each sub-instance.

```
instance = o.run_xflow('AppendID', 'algo_public', {'inputTableName': 'input_table', 'outputTableName': 'output_table'})
for sub_inst_name, sub_inst in o.get_xflow_sub_instances(instance).items():
    print('%s: %s' % (sub_inst_name, sub_inst.get_logview_address()))
```

Obtain the status of an instance

An instance can be in the `Running`, `Suspended`, or `Terminated` state. You can use the following methods to obtain the status of an instance:

- `status` : obtains the status of an instance.
- `is_terminated` : checks whether the execution of the current instance is completed.
- `is_successful` : checks whether the execution of the current instance is successful. If the instance is running or fails, `False` is returned.

```
instance = o.get_instance('2016042605520945g9k5pvyi2')
instance.status
<Status.TERMINATED: 'Terminated'>
from odps.models import Instance
instance.status == Instance.Status.TERMINATED
True
instance.status.value
'Terminated'
```

 **Note** You can call the `wait_for_completion` method to ask the system to return the status until the execution of the instance is completed. The `wait_for_success` method functions similarly. The difference is that if the instance fails, an exception is thrown.

Perform operations on sub-instances

A running instance may contain one or more sub-instances. You can call the following methods to perform specific operations on sub-instances:

- `get_task_names` : obtains all sub-instances. This method returns the names of all sub-instances.

```
instance.get_task_names()
['SQLDropTableTask']
```

- `get_task_result` : obtains the execution result of a specific sub-instance. This method returns the execution result of each sub-instance in the form of a dictionary.

```
instance = o.execute_sql('select * from pyodps_iris limit 1')
instance.get_task_names()
['AnonymousSQLTask']
instance.get_task_result('AnonymousSQLTask')
"sepallength","sepalwidth","petallength","petalwidth","name"\n5.1,3.5,1.4,0.2,"Iris-setosa"\n'
instance.get_task_results()
OrderedDict([(('AnonymousSQLTask', "sepallength","sepalwidth","petallength","petalwidth","name"\n5.1,3.5,1.4,0.2,"Iris-setosa"\n)])])
```

- `get_task_progress` : obtains the current running progress of a specific sub-instance when the instance is running.

```
while not instance.is_terminated():
    for task_name in instance.get_task_names():
        print(instance.id, instance.get_task_progress(task_name).get_stage_progress_formatted_string())
        time.sleep(10)
20160519101349613gzbzufck2 2016-05-19 18:14:03 M1_Stg1_job0:0/1/1[100%]
```

6.4.6. Resources

PyODPS supports file and table resources. This topic describes the common operations on resources in PyODPS.

resource list_resources

Basic operations

- `list_resources()` : lists all resources in a specific project.
- `exist_resource()` : checks whether a specific resource exists.
- `delete_resource()` : deletes a specific resource. You can also call the `drop()` method to delete a resource.
- `create_resource()` : creates a resource.
- `open_resource()` : reads a resource.

Manage file resources

Files of basic file types and the .py, .jar, and .archive types are supported.

- Create a file resource

To create a file resource, you can call the `create_resource()` method with a resource name, a file type, and a file-like object or string specified.

```
resource = o.create_resource('test_file_resource', 'file', file_obj=open('/to/path/file')) # Create a file resource with a file-like object.
resource = o.create_resource('test_py_resource', 'py', file_obj='import this') # Create a file resource with a string.
```

- Read and modify a file resource

You can use one of the following methods to open a file resource:

- Call the `open()` method for the file resource.
- Call the `open_resource()` method of a MaxCompute entry object.

An opened file resource is a file-like object. Similar to the `open()` method that is built in Python, the preceding two methods also allow you to specify the opening mode of a file resource.

```
with resource.open('r') as fp: # Open the file in read mode.
    content = fp.read() # Read all content of the file.
    fp.seek(0) # Move the file pointer to the beginning of the file.
    lines = fp.readlines() # Read multiple lines.
    fp.write('Hello World') # Error. Data cannot be written to a file in read mode.

with o.open_resource('test_file_resource', mode='r+') as fp: # Open the file in read/write mode.
    fp.read()
    fp.tell() # Locate the current position of the file pointer.
    fp.seek(10)
    fp.truncate() # Truncate the file to the specified length.
    fp.writelines(['Hello\n', 'World\n']) # Write multiple lines to the file.
    fp.write('Hello World')
    fp.flush() # Manually call the method to submit the update to MaxCompute.
```

PyODPS supports the following file opening modes:

- `r` : read mode. The file can be opened but data cannot be written to it.
- `w` : write mode. Data can be written to the file, but data in the file cannot be read. If a file is opened in write mode, the file content is cleared first.
- `a` : append mode. Data can be added to the end of the file.
- `r+` : read/write mode. You can read data from and write data to the file.

- `w+` : This mode is similar to `r+` , but the file content is cleared first when the file is opened.
- `a+` : This mode is similar to `r+` , but data can be written only to the end of the file.

In PyODPS, you can also open a file resource in binary mode. For example, some compressed files must be opened in binary mode.

- `rb` : binary read mode
- `r+b` : binary read/write mode

Table resources

- Create a table resource

```
o.create_resource('test_table_resource', 'table', table_name='my_table', partition='pt=test')
```

- Update a table resource

```
table_resource = o.get_resource('test_table_resource')
table_resource.update(partition='pt=test2', project_name='my_project2')
```

- Obtain a table and a partition of the table

```
table_resource = o.get_resource('test_table_resource')
table = table_resource.table
print(table.name)
partition = table_resource.partition
print(partition.spec)
```

- Read data from and write data to a table

```
table_resource = o.get_resource('test_table_resource')
with table_resource.open_writer() as writer:
    writer.write([0, 'aaaa'])
    writer.write([1, 'bbbb'])
with table_resource.open_reader() as reader:
    for rec in reader:
        print(rec)
```

6.4.7. Functions

You can create user defined functions (UDFs) and use them in MaxCompute SQL.

Basic operations

- `list_functions()` : obtains all functions in a specific project.
- `exist_function()` : checks whether a specific function exists.

- `get_function()` : obtains a specific function.
- `create_function()` : creates a function.
- `delete_function()` : deletes a specific function.

Create a function

You can call the `create_function()` method of a MaxCompute entry object to create a function. The following code shows examples of how to create a function by using a resource in the current project and in another project:

```
# Reference a resource in the current project.
resource = o.get_resource('my_udf.py')
function = o.create_function('test_function', class_type='my_udf.Test', resources=[resource])

# Reference a resource in another project.
resource2 = o.get_resource('my_udf.py', project='another_project')
function2 = o.create_function('test_function2', class_type='my_udf.Test', resources=[resource2])
```

Delete a function

You can call the `delete_function()` method of a MaxCompute entry object to delete a function. You can also call the `drop()` method to delete a function. The following code shows examples of how to delete a function by using the `delete_function()` and `drop()` methods:

```
o.delete_function('test_function')
function.drop() # Call the drop() method if the function exists.
```

Update a function

You can call the `update()` method to update a function. The following code shows an example:

```
function = o.get_function('test_function')
new_resource = o.get_resource('my_udf2.py')
function.class_type = 'my_udf2.Test'
function.resources = [new_resource, ]
function.update() # Update the function.
```

6.5. Configurations

This topic describes the configuration options provided by PyODPS.

general configuration data upload and download configuration DataFrame configuration
machine learning configuration

You can use `odps.options` to obtain the configuration options provided by PyODPS.

```

from odps import options
# Set the lifecycle option to specify the lifecycle of all output tables.
options.lifecycle = 30
# Set the tunnel.string_as_binary option to True to use bytes instead of Unicode to download data of
the STRING type.
options.tunnel.string_as_binary = True
# When you execute PyODPS DataFrames in MaxCompute, you can refer to the following configuration
to set the limit to a relatively large value during a sort operation.
options.df.odps.sort.limit = 100000000

```

General configurations

Option	Description	Default value
end_point	The endpoint of MaxCompute.	None
default_project	The default project.	None
log_view_host	The hostname of Logview.	None
log_view_hours	The retention time of Logview. Unit: hours.	24
local_timezone	The time zone that is used. True indicates local time, and False indicates UTC. The time zone of <code>pytz</code> can also be used.	None
lifecycle	The lifecycle of all tables.	None
temp_lifecycle	The lifecycle of temporary tables.	1
biz_id	The user ID.	None
verbose	Specifies whether to display logs.	False
verbose_log	The log receiver.	None
chunk_size	The size of the write buffer.	1496
retry_times	The number of request retries.	4
pool_connections	The number of cached connections in the connection pool.	10

Option	Description	Default value
pool_maxsize	The maximum capacity of the connection pool.	10
connect_timeout	The connection timeout period.	5
read_timeout	The read timeout period.	120
api_proxy	The API proxy server.	None
data_proxy	The data proxy server.	None
completion_size	The limit on the number of object completion listing items.	10
notebook_repr_widget	Specifies whether to use interactive graphs.	True
sql.settings	Global hints for MaxCompute SQL.	None
sql.use_odps2_extension	Specifies whether to enable MaxCompute 2.0 language extension.	False

Data upload and download configurations

Option	Description	Default value
tunnel.endpoint	The endpoint of MaxCompute Tunnel.	None
tunnel.use_instance_tunnel	Specifies whether to use InstanceTunnel to obtain execution results.	True
tunnel.limit_instance_tunnel	Specifies whether to limit the number of data records obtained by using InstanceTunnel.	None
tunnel.string_as_binary	Specifies whether to use bytes instead of Unicode for data of the STRING type.	False

DataFrame configurations

Option	Description	Default value
interactive	Specifies whether DataFrames are used in an interactive environment.	Depends on the detection value.

Option	Description	Default value
df.analyze	Specifies whether to enable functions that are not built in MaxCompute.	True
df.optimize	Specifies whether to enable full DataFrame optimization.	True
df.optimizes.pp	Specifies whether to enable DataFrame predicate pushdown optimization.	True
df.optimizes.cp	Specifies whether to enable DataFrame column pruning optimization.	True
df.optimizes.tunnel	Specifies whether to enable DataFrame tunnel optimization.	True
df.quote	Specifies whether to use a pair of grave accents (``) to mark field and table names in the backend of MaxCompute SQL.	True
df.libraries	The resource name of the third-party library that is used for DataFrame operations.	None
df.supersede_libraries	Specifies whether to use the self-uploaded NumPy to replace the version in the service.	False
df.odps.sort.limit	The default limit on the number of items that are added during a sort operation of DataFrames.	10000

Machine learning configurations

Option	Description	Default value
ml.xflow_settings	The XFlow execution configuration.	None
ml.xflow_project	The default XFlow project name.	algo_public
ml.use_model_transfer	Specifies whether to use ModelTransfer to obtain the Predictive Model Markup Language (PMML) files of models.	False

Option	Description	Default value
ml.model_volume	The name of the volume used by ModelTransfer.	pyodps_volume

6.6. DataFrame

6.6.1. Quick start

This topic describes how to create and manage a DataFrame object. It also provides information about how to use DataFrame to process data.

Create and manage a DataFrame object

The **MovieLens 100K** dataset is used in this example. The following tables exist:

pyodps_ml_100k_movies (movie-related data), pyodps_ml_100k_users (user-related data), and pyodps_ml_100k_ratings (rating-related data).

1. If no MaxCompute objects are available in the runtime environment, create a MaxCompute object.

```
from odps import ODPS
o = ODPS(**your-access-id**, **your-secret-access-key**,
        project=**your-project**, endpoint=**your-end-point**)
```

2. Create a DataFrame object by specifying a table.

```
from odps.df import DataFrame
users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

3. Use the `dtypes` attribute to view the fields of the DataFrame object and the data types of the fields.

```
users.dtypes
odps.Schema {
  user_id      int64
  age          int64
  sex          string
  occupation   string
  zip_code     string
}
```

4. Use the `head` method to have a quick preview on the first N data records.

```
users.head(10)
```

	user_id	age	sex	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213
5	6	42	M	executive	98101
6	7	57	M	administrator	91344
7	8	36	M	administrator	05201
8	9	29	M	student	01002
9	10	53	M	lawyer	90703

5. If you do not want to view all the fields, perform the following operations as needed:

- Specify the fields that you want to query.

```
users[['user_id', 'age']].head(5)
```

	user_id	age
0	1	24
1	2	53
2	3	23
3	4	24
4	5	33

- Exclude the fields that you do not want to query.

```
users.exclude('zip_code', 'age').head(5)
```

	user_id	sex	occupation
0	1	M	technician
1	2	F	other
2	3	M	writer
3	4	M	technician
4	5	F	other

- Exclude some fields and add new fields based on computation. For example, add the `sex_bool` attribute and set it to True if `sex` is `M`. Otherwise, set it to False.

```
users.select(users.exclude('zip_code', 'sex'), sex_bool=users.sex == 'M').head(5)
```

	user_id	age	occupation	sex_bool
0	1	24	technician	True
1	2	53	other	False
2	3	23	writer	True
3	4	24	technician	True
4	5	33	other	False

6. Obtain the numbers of male and female users.

```
users.groupby(users.sex).agg(count=users.count())
```

	sex	count
0	F	273
1	M	670

7. Divide users by occupation, obtain the first 10 occupations that have the largest population, and sort the occupations in descending order of population.

```
df = users.groupby('occupation').agg(count=users['occupation'].count())
df.sort(df['count'], ascending=False)[:10]
```

	occupation	count
0	student	196
1	other	105
2	educator	95
3	administrator	79
4	engineer	67
5	programmer	66
6	librarian	51
7	writer	45
8	executive	32
9	scientist	31

Alternatively, use the `value_counts` method. Note that the number of records returned by this method is limited by the `options.df.odps.sort.limit` parameter. For more information, see [Configurations](#).

```
users.occupation.value_counts()[:10]
  occupation count
0   student   196
1    other   105
2  educator    95
3 administrator    79
4   engineer    67
5  programmer    66
6   librarian    51
7    writer    45
8  executive    32
9   scientist    31
```

8. Show data in an intuitive graph.

```
%matplotlib inline
```

9. Use a horizontal column chart to visualize the data.

```
users['occupation'].value_counts().plot(kind='barh', x='occupation', ylabel='profession')
<matplotlib.axes._subplots.AxesSubplot at 0x10653cfd0>
```

□

10. Divide users into 30 groups by age and view the histogram of age distribution.

```
users.age.hist(bins=30, title="Distribution of users' ages", xlabel='age', ylabel='count of users')
<matplotlib.axes._subplots.AxesSubplot at 0x10667a510>
```

□

11. Join the three tables and save them as a new table.

```

movies = DataFrame(o.get_table('pyodps_ml_100k_movies'))
ratings = DataFrame(o.get_table('pyodps_ml_100k_ratings'))

o.delete_table('pyodps_ml_100k_lens', if_exists=True)
lens = movies.join(ratings).join(users).persist('pyodps_ml_100k_lens')

lens.dtypes
odps.Schema {
  movie_id          int64
  title             string
  release_date      string
  video_release_date string
  imdb_url          string
  user_id           int64
  rating            int64
  unix_timestamp    int64
  age               int64
  sex              string
  occupation        string
  zip_code          string
}

```

12. Divide users aged 0 to 80 into eight age groups.

```

labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-69', '70-79']
cut_lens = lens[lens, lens.age.cut(range(0, 81, 10), right=False, labels=labels).rename('age_group')]

```

13. View the first 10 ages each with only one user, as well as their age groups.

```

cut_lens['age_group', 'age'].distinct()[:10]
age_group age
0  0-9  7
1  10-19 10
2  10-19 11
3  10-19 13
4  10-19 14
5  10-19 15
6  10-19 16
7  10-19 17
8  10-19 18
9  10-19 19

```

14. View the total rating and average rating of users in each age group.

```
cut_lens.groupby('age_group').agg(cut_lens.rating.count().rename('total_rating'), cut_lens.rating.mean().rename('avg_rating'))
```

	age_group	avg_rating	total_rating
0	0-9	3.767442	43
1	10-19	3.486126	8181
2	20-29	3.467333	39535
3	30-39	3.554444	25696
4	40-49	3.591772	15021
5	50-59	3.635800	8704
6	60-69	3.648875	2623
7	70-79	3.649746	197

Use DataFrame to process data

Before you perform the following steps, [download the Iris dataset](#) and save the file to your computer as sample data. A DataWorks PyODPS node is used in this example. For more information, see [PyODPS node](#).

1. Create a test data table.

Use the table management feature of DataWorks to create a table. Then, click **DDL mode**.

□

Execute the following statement to generate a table structure:

```
CREATE TABLE pyodps_iris (
  sepallength double COMMENT 'sepallength(cm)',
  sepalwidth double COMMENT 'sepalwidth(cm)',
  petallength double COMMENT 'petallength(cm)',
  petalwidth double COMMENT 'petalwidth(cm)',
  name string COMMENT 'name'
);
```

2. Upload test data.

Click **Import data**.

□

Enter the table name, upload the downloaded dataset, and then click **Next**.

□

Select **Match by location** and click **Import Data**.

□

3. Create a PyODPS node to store and run code.

□

4. Enter the code and click the Run icon. You can view the result in the Run Log section in the lower pane.

□

Code details:


```

from odps.df import DataFrame
from odps.df import output

iris = DataFrame(o.get_table('pyodps_iris')) # Create the DataFrame object iris from the MaxCompute table.

print iris.head(10)
print iris.sepalength.head(5) # Display part of the iris content.

# Use a UDF to calculate the sum of two columns of iris.
print iris.apply(lambda row: row.sepalength + row.sepalwidth, axis=1, reduce=True, types='float')
        .rename('sepaladd').head(3)

# Specify the name and type of the output field for the UDF.
@output(['iris_add', 'iris_sub'], ['float', 'float'])
def handle(row):
    # Use the yield keyword to return multiple rows of results.
    yield row.sepalength - row.sepalwidth, row.sepalength + row.sepalwidth
    yield row.petalength - row.petalwidth, row.petalength + row.petalwidth
# Display the results of the first five rows. axis=1 indicates that the axis of the column extends horizontally.
print iris.apply(handle,axis=1).head(5)

```

6.6.2. Create a DataFrame object

This topic describes how to create a DataFrame object. After this object is created, it can be used to reference a data source.

Background information

To use a DataFrame project, you must familiarize yourself with the operations on the `Collection` (`DataFrame`), `Sequence`, and `Scalar` objects. `Collection` indicates a tabular data structure (two-dimensional structure). `Sequence` indicates a column (one-dimensional structure). `Scalar` indicates a scalar object.

If you use Pandas data to create a DataFrame object, the object contains the actual data. If you use a MaxCompute table to create a DataFrame object, the object does not contain the actual data. The object contains only data operations. After a DataFrame object is created, MaxCompute can store and compute data.

Procedure

DataFrame object is the only Collection object that you must create. A DataFrame object can be used to reference MaxCompute tables, MaxCompute partitions, Pandas DataFrame objects, and SQLAlchemy tables (database tables). The reference operations for these data sources are the same. You can process data without the need to modify code. You only need to change the input and output pointers. This way, you can migrate small amounts of test code that is running locally to MaxCompute. The accuracy of the migration is ensured by PyODPS DataFrame.

To create a DataFrame object, you only need to call the required MaxCompute table, Pandas DataFrame object, or SQLAlchemy table.

```
from odps.df import DataFrame

# Create a DataFrame object by using a MaxCompute table.
iris = DataFrame(o.get_table('pyodps_iris'))
iris2 = o.get_table('pyodps_iris').to_df() # Use the to_df method for the table.

# Create a DataFrame object by using a MaxCompute partition.
pt_df = DataFrame(o.get_table('partitioned_table').get_partition('pt=20171111'))
pt_df2 = o.get_table('partitioned_table').get_partition('pt=20171111').to_df() # Use the to_df method for the partition.

# Create a DataFrame object by using a Pandas DataFrame object.
import pandas as pd
import numpy as np
df = DataFrame(pd.DataFrame(np.arange(9).reshape(3, 3), columns=list('abc'))))

# Create a DataFrame object by using an SQLAlchemy table.
engine = sqlalchemy.create_engine('mysql://root:123456@localhost/movielens')
metadata = sqlalchemy.MetaData(bind = engine) # Bind the metadata to a database engine.
table = sqlalchemy.Table('top_users', metadata, extend_existing=True, autoload=True)
users = DataFrame(table)
```

If you create a DataFrame object by using a Pandas DataFrame object, note the following points during the initialization of the DataFrame object:

- PyODPS DataFrame attempts to infer the NUMPY OBJECT or STRING data type. If a column is empty, an error is returned. To avoid these errors, set `unknown_as_string` to True, and convert the data type of this column to STRING.
- You can use the `as_type` parameter to forcibly convert the data type. If a basic data type is used, this type is forcibly converted when you create a PyODPS DataFrame object. If a Pandas DataFrame object contains a LIST or DICT column, the system does not infer the data type of this column. You must manually set `as_type` to DICT.

```
df2 = DataFrame(df, unknown_as_string=True, as_type={'null_col2': 'float'})
df2.dtypes
odps.Schema {
  sepallength      float64
  sepalwidth       float64
  petallength      float64
  petalwidth       float64
  name             string
  null_col1        string # The data type cannot be identified. You can set unknown_as_string to True
to convert the data type to STRING.
  null_col2        float64 # The data type is forcibly converted to FLOAT.
}
df4 = DataFrame(df3, as_type={'list_col': 'list<int64>'})
df4.dtypes
odps.Schema {
  id      int64
  list_col list<int64> # The data type cannot be identified or automatically converted. You must specify
as_type.
}
```

 **Note** PyODPS DataFrame does not allow you to upload Object Storage Service (OSS) or Tablestore (OTS) external tables to MaxCompute.

6.6.3. Sequence

SequenceExpr represents a column in a two-dimensional dataset. You cannot manually create a SequenceExpr object. Instead, you can retrieve one from a Collection object.

Column retrieval

You can use `collection.column_name` to retrieve a column.

```
iris.sepallength.head(5)
sepallength
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
```

If the column name is stored in a string variable, you can use `df[column_name]` to retrieve a

column.

```
iris['sepalength'].head(5)
sepalength
0      5.1
1      4.9
2      4.7
3      4.6
4      5.0
```

Column types

DataFrame has its own type system. When a table is initialized, the MaxCompute data types are cast. This design provides support for more types of execution backends. The execution backend of DataFrame can be MaxCompute SQL, Pandas, or a MySQL or PostgreSQL database.

The following table lists the mappings between MaxCompute and DataFrame data types.

MaxCompute type	DataFrame type
BIGINT	INT64
DOUBLE	FLOAT64
STRING	STRING
DATETIME	DATETIME
BOOLEAN	BOOLEAN
DECIMAL	DECIMAL
ARRAY<VALUE_TYPE>	LIST<VALUE_TYPE>
MAP<KEY_TYPE, VALUE_TYPE>	DICT<KEY_TYPE, VALUE_TYPE>
When <code>options.sql.use_odps2_extension=True</code> , the following data types are also supported.	
TINYINT	INT8
SMALLINT	INT16
INT	INT32
FLOAT	FLOAT32

The following items describe DataFrame data types:

- PyODPS DataFrame supports the following data types: INT8, INT16, INT32, INT64, FLOAT32, FLOAT64, BOOLEAN, STRING, DECIMAL, DATETIME, LIST, and DICT.
- For LIST and DICT types, you must specify the types of the values they contain. Otherwise, an error occurs.

- DataFrame does not support the TIMESTAMP and STRUCT types introduced in MaxCompute 2.0. The new types will be supported in future releases.
- You can use `sequence.dtype` to retrieve the data type of a Sequence object.

```
iris.sepalength.dtype
float64
```

- You can use the `astype` method to change the type of a column. This method requires a type as input and returns the converted Sequence object as output.

```
iris.sepalength.astype('int')
sepalength
0      5
1      4
2      4
3      4
4      5
```

Column name

In DataFrame computing, a Sequence object must have a column name. Typically, DataFrame automatically creates a name for each Sequence object.

```
iris.groupby('name').sepalwidth.max()
sepalwidth_max
0      4.4
1      3.4
2      3.8
```

In the preceding example, `sepalwidth` is named `sepalwidth_max` after the `max()` method is applied to take the maximum value. In some operations, such as adding a Scalar object to a Sequence object, the resulting column is automatically assigned the name of the Sequence object. In other cases, you need to manually name a Sequence object.

You can use the `rename` method to rename a Sequence object.

```
iris.sepalwidth.rename('sepal_width').head(5)
sepal_width
0      3.5
1      3.0
2      3.2
3      3.1
4      3.6
```

Simple column transformations

You can perform operations on a Sequence object to obtain a new Sequence object. This is similar to performing operations on a simple Python variable.

```
(iris.sepalength + 5).head(5)
  sepalength
0      10.1
1       9.9
2       9.7
3       9.6
4      10.0
```

Note that when two columns are involved in operations, you need to manually name the resulting column. For more information about column transformations, see [Column operations](#).

```
(iris.sepalength + iris.sepalwidth).rename('sum_sepal').head(5)
  sum_sepal
0       8.6
1       7.9
2       7.9
3       7.7
4       8.6
```

6.6.4. Collection

CollectionExpr supports all operations on DataFrame two-dimensional datasets. It can be seen as a MaxCompute table or a spreadsheet. DataFrame objects are also CollectionExpr objects. CollectionExpr supports various operations on two-dimensional datasets such as column operations, filter operations, and transformation operations.

Retrieve types

You can use the `dtypes` method to retrieve the types of all columns in a CollectionExpr object. The `dtypes` method returns a [Schema type](#).

```
iris.dtypes
odps.Schema {
  sepalength      float64
  sepalwidth      float64
  petallength     float64
  petalwidth      float64
  name            string
}
```

Select, add, and remove columns

- Select columns

You can use the `expr[columns]` syntax to select certain columns from a `CollectionExpr` object to form a new dataset.

```
iris['name', 'sepallength'].head(5)
```

	name	sepallength
0	Iris-setosa	5.1
1	Iris-setosa	4.9
2	Iris-setosa	4.7
3	Iris-setosa	4.6
4	Iris-setosa	5.0

Note

If only one column is needed, you need to add a comma (,) after the column name or explicitly mark the column as a list, such as `iris[iris.sepal_length,]` or `iris[[iris.sepal_length]]`. Otherwise, a Sequence object, instead of a Collection object, is returned.

- Delete columns

You can use the `exclude` method to exclude certain columns of the original dataset from the new dataset.

```
iris.exclude('sepallength', 'petallength')[5]
```

	sepalwidth	petalwidth	name
0	3.5	0.2	Iris-setosa
1	3.0	0.2	Iris-setosa
2	3.2	0.2	Iris-setosa
3	3.1	0.2	Iris-setosa
4	3.6	0.2	Iris-setosa

In PyODPS version 0.7.2 and later, you can use a new method to directly exclude certain columns from the dataset.

```
del iris['sepallength']
del iris['petallength']
iris[:5]
```

	sepalwidth	petalwidth	name
0	3.5	0.2	Iris-setosa
1	3.0	0.2	Iris-setosa
2	3.2	0.2	Iris-setosa
3	3.1	0.2	Iris-setosa
4	3.6	0.2	Iris-setosa

- Add columns

You can use the `expr[expr, new_sequence]` syntax to add a transformed column to an existing Collection object. The new column is part of the new Collection.

In the following example, a new column is created by adding one to each element in the `sepalwidth` column of `iris`, renamed to `sepalwidthplus1`, and appended to the dataset to form a new dataset.

```
iris[iris, (iris.sepalwidth + 1).rename('sepalwidthplus1')].head(5)
```

	sepallength	sepalwidth	petallength	petalwidth	name \
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa


```
sepalwidthplus1
```

0	4.5
1	4.0
2	4.2
3	4.1
4	4.6

When you use the `expr[expr, new_sequence]` syntax, note that the transformed column may have the same name as the original column. Rename the new column if you want to merge it with the original Collection.

In PyODPS version 0.7.2 and later, you can directly append a column to the current dataset.


```
iris['sepalwidthplus1'] = iris.sepalwidth + 1
iris.head(5)
```

	sepalwidth	sepalwidthplus1	petallength	petalwidth	name \
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa


```
iris['sepalwidthplus1']
```

	sepalwidthplus1
0	4.5
1	4.0
2	4.2
3	4.1
4	4.6

- Add and delete columns simultaneously

You can also use the `exclude` method to exclude the original column and append the transformed column to the dataset, without the need to rename the new column.

```
iris[iris.exclude('sepalwidth'), iris.sepalwidth * 2].head(5)
```

	sepalwidth	sepalwidth * 2	petallength	petalwidth	name
0	5.1	7.0	1.4	0.2	Iris-setosa
1	4.9	6.0	1.4	0.2	Iris-setosa
2	4.7	6.4	1.3	0.2	Iris-setosa
3	4.6	6.2	1.5	0.2	Iris-setosa
4	5.0	7.2	1.4	0.2	Iris-setosa

In PyODPS version 0.7.2 and later, you can directly overwrite an existing column on the current dataset. The following example illustrates the syntax.

```
iris['sepalwidth'] = iris.sepalwidth * 2
iris.head(5)
```

	sepalwidth	sepalwidth * 2	petallength	petalwidth	name
0	5.1	7.0	1.4	0.2	Iris-setosa
1	4.9	6.0	1.4	0.2	Iris-setosa
2	4.7	6.4	1.3	0.2	Iris-setosa
3	4.6	6.2	1.5	0.2	Iris-setosa
4	5.0	7.2	1.4	0.2	Iris-setosa

The `select` method provides another way to create a new Collection. When you use this method, you need to pass in the selected columns as input parameters. You can use the `keyword` argument to rename a column to the given keyword.

```
iris.select('name', sepalwidthminus1=iris.sepalwidth - 1).head(5)
```

	name	sepalwidthminus1
0	Iris-setosa	2.5
1	Iris-setosa	2.0
2	Iris-setosa	2.2
3	Iris-setosa	2.1
4	Iris-setosa	2.6

You can also pass in a lambda expression, which takes the result from the previous operation as a parameter. During the execution, PyODPS checks the lambda expression and passes in the Collection object generated from the previous operation and replaces it with the correct columns.

```
iris['name', 'petallength'][(lambda x: x.name)].head(5)
```

	name
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa

In PyODPS version 0.7.2 and later, conditional assignments are supported.

```
iris[iris.sepalwidth > 5.0, 'sepalwidth'] = iris.sepalwidth * 2
```

```
iris.head(5)
```

	sepalwidth	sepalwidth	petallength	petalwidth	name
0	5.1	14.0	1.4	0.2	Iris-setosa
1	4.9	6.0	1.4	0.2	Iris-setosa
2	4.7	6.4	1.3	0.2	Iris-setosa
3	4.6	6.2	1.5	0.2	Iris-setosa
4	5.0	7.2	1.4	0.2	Iris-setosa

Introduce constants and random numbers

- DataFrame allows you to append a column of constants to a Collection object. **Scalar** is required for this operation, and you need to manually specify the column name.

```
from odps.df import Scalar
iris[iris, Scalar(1).rename('id')][:5]
```

	sepalength	sepalwidth	petallength	petalwidth	name	id
0	5.1	3.5	1.4	0.2	Iris-setosa	1
1	4.9	3.0	1.4	0.2	Iris-setosa	1
2	4.7	3.2	1.3	0.2	Iris-setosa	1
3	4.6	3.1	1.5	0.2	Iris-setosa	1
4	5.0	3.6	1.4	0.2	Iris-setosa	1

You can use **NullScalar** to specify a null column. In this case, you need to specify the field type.

```
from odps.df import NullScalar
iris[iris, NullScalar('float').rename('fid')][:5]
```

	sepal_length	sepal_width	petal_length	petal_width	category	fid
0	5.1	3.5	1.4	0.2	Iris-setosa	None
1	4.9	3.0	1.4	0.2	Iris-setosa	None
2	4.7	3.2	1.3	0.2	Iris-setosa	None
3	4.6	3.1	1.5	0.2	Iris-setosa	None
4	5.0	3.6	1.4	0.2	Iris-setosa	None

In PyODPS version 0.7.12 and later, a simpler method is provided.

```
iris['id'] = 1
iris
```

	sepalength	sepalwidth	petallength	petalwidth	name	id
0	5.1	3.5	1.4	0.2	Iris-setosa	1
1	4.9	3.0	1.4	0.2	Iris-setosa	1
2	4.7	3.2	1.3	0.2	Iris-setosa	1
3	4.6	3.1	1.5	0.2	Iris-setosa	1
4	5.0	3.6	1.4	0.2	Iris-setosa	1

Note that this method cannot automatically recognize the type of null values. Therefore, we recommend that you use the following syntax to add null columns.

```
iris['null_col'] = NullScalar('float')
iris
```

	sepalength	sepalwidth	petallength	petalwidth	name	null_col
0	5.1	3.5	1.4	0.2	Iris-setosa	None
1	4.9	3.0	1.4	0.2	Iris-setosa	None
2	4.7	3.2	1.3	0.2	Iris-setosa	None
3	4.6	3.1	1.5	0.2	Iris-setosa	None
4	5.0	3.6	1.4	0.2	Iris-setosa	None

- DataFrame also allows you to append a column of random numbers to a Collection object. The column type is FLOAT and the value range is 0-1. Each number has a different value. **RandomScalar** is required for this operation, and its parameter is an optional random seed.

```
from odps.df import RandomScalar
iris[iris, RandomScalar().rename('rand_val')][:5]
```

	sepallength	sepalwidth	petallength	petalwidth	name	rand_val
0	5.1	3.5	1.4	0.2	Iris-setosa	0.000471
1	4.9	3.0	1.4	0.2	Iris-setosa	0.799520
2	4.7	3.2	1.3	0.2	Iris-setosa	0.834609
3	4.6	3.1	1.5	0.2	Iris-setosa	0.106921
4	5.0	3.6	1.4	0.2	Iris-setosa	0.763442

Filter data

A Collection object allows you to filter data. In the following example, the records where **sepallength** is greater than 5 are retrieved.

```
iris[iris.sepallength > 5].head(5)
```

	sepallength	sepalwidth	petallength	petalwidth	name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	5.4	3.9	1.7	0.4	Iris-setosa
2	5.4	3.7	1.5	0.2	Iris-setosa
3	5.8	4.0	1.2	0.2	Iris-setosa
4	5.7	4.4	1.5	0.4	Iris-setosa

The following example shows how to query records by using multiple filter conditions:

- AND (&) operator

```
iris[(iris.sepallength < 5) & (iris['petallength'] > 1.5)].head(5)
```

	sepallength	sepalwidth	petallength	petalwidth	name
0	4.8	3.4	1.6	0.2	Iris-setosa
1	4.8	3.4	1.9	0.2	Iris-setosa
2	4.7	3.2	1.6	0.2	Iris-setosa
3	4.8	3.1	1.6	0.2	Iris-setosa
4	4.9	2.4	3.3	1.0	Iris-versicolor

- OR (|) operator

```
iris[(iris.sepalwidth < 2.5) | (iris.sepalwidth > 4)].head(5)
```

	sepalwidth	sepalwidth	petallength	petalwidth	name
0	5.7	4.4	1.5	0.4	Iris-setosa
1	5.2	4.1	1.5	0.1	Iris-setosa
2	5.5	4.2	1.4	0.2	Iris-setosa
3	4.5	2.3	1.3	0.3	Iris-setosa
4	5.5	2.3	4.0	1.3	Iris-versicolor

 **Note** You must use an ampersand `&` to represent the AND operator and use a vertical bar `|` to represent the OR operator. `and` and `or` are not allowed.

- NOT (~) operator

```
iris[~(iris.sepalwidth > 3)].head(5)
```

	sepalwidth	sepalwidth	petallength	petalwidth	name
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.4	2.9	1.4	0.2	Iris-setosa
2	4.8	3.0	1.4	0.1	Iris-setosa
3	4.3	3.0	1.1	0.1	Iris-setosa
4	5.0	3.0	1.6	0.2	Iris-setosa

- You can explicitly call the `filter` method to specify multiple conditions by using AND (&) operators.

```
iris.filter(iris.sepalwidth > 3.5, iris.sepalwidth < 4).head(5)
```

	sepalwidth	sepalwidth	petallength	petalwidth	name
0	5.0	3.6	1.4	0.2	Iris-setosa
1	5.4	3.9	1.7	0.4	Iris-setosa
2	5.4	3.7	1.5	0.2	Iris-setosa
3	5.4	3.9	1.3	0.4	Iris-setosa
4	5.7	3.8	1.7	0.3	Iris-setosa

- You can use a lambda expression to perform sequential operations.

```
iris[iris.sepalwidth > 3.8]['name', lambda x: x.sepalwidth + 1]
```

	name	sepalwidth
0	Iris-setosa	6.4
1	Iris-setosa	6.8
2	Iris-setosa	6.7
3	Iris-setosa	6.4
4	Iris-setosa	6.2
5	Iris-setosa	6.5

- If a Collection object contains a BOOLEAN column, you can use the column as a filter condition.

```
df.dtypes
odps.Schema {
  a boolean
  b int64
}
df[df.a]
   a b
0 True 1
1 True 3
```

Therefore, when you retrieve a single Sequence from a Collection object, only the BOOLEAN column can be used as a valid filter condition.

```
df[df.a, ] # retrieve a one-column collection.
df[[df.a]] # retrieve a one-column collection.
df.select(df.a) # retrieve a one-column collection explicitly.
df[df.a] # column 'a' is a boolean column. Filter data on the boolean column.
df.a # retrieve a column from a collection.
df['a'] # retrieve a column from a collection.
```

- You can also use the `query` method in Pandas to filter data, and use column names such as `sepalength` in an expression. In query statements, both the ampersand `&` and `and` represent the AND operator, and both the vertical bar `|` and `or` represent the OR operator.

```
iris.query("(sepalength < 5) and (petallength > 1.5)").head(5)
```

	sepalength	sepalwidth	petallength	petalwidth	name
0	4.8	3.4	1.6	0.2	Iris-setosa
1	4.8	3.4	1.9	0.2	Iris-setosa
2	4.7	3.2	1.6	0.2	Iris-setosa
3	4.8	3.1	1.6	0.2	Iris-setosa
4	4.9	2.4	3.3	1.0	Iris-versicolor

When a local variable is required in an expression, add an at sign `@` before the variable name.

```
var = 4
iris.query("(iris.sepalwidth < 2.5) | (sepalwidth > @var)").head(5)
```

	sepalwidth	sepalwidth	petalwidth	petalwidth	name
0	5.7	4.4	1.5	0.4	Iris-setosa
1	5.2	4.1	1.5	0.1	Iris-setosa
2	5.5	4.2	1.4	0.2	Iris-setosa
3	4.5	2.3	1.3	0.3	Iris-setosa
4	5.5	2.3	4.0	1.3	Iris-versicolor

The `query` method supports the following syntax.

Syntax	Description
name	Names without the at sign <code>@</code> are processed as column names. Names with the at sign (<code>@</code>) are processed as local variables.
operator	The following operators are supported: <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>//</code> , <code>%</code> , <code>**</code> , <code>==</code> , <code>!=</code> , <code><</code> , <code><=</code> , <code>></code> , <code>>=</code> , <code>in</code> , <code>not in</code> .
bool	An ampersand <code>&</code> and <code>and</code> represent the AND operator. A vertical bar <code> </code> and <code>or</code> represent the OR operator.
attribute	This syntax is used to retrieve the attribute of the object.
index, slice, subscript	Slice operations.

Lateral view

You can use the `explode` method to convert the LIST and MAP columns into multiple rows for output. You can also use the `apply` method to generate a lateral view. You often need to merge the output with the columns in the original table for operations such as aggregation. In this case, you can use the lateral view function of DataFrame. That is, map the collection generated by the lateral view function with the column names in the original collection.

The following example shows how to generate a lateral view.

```
df
  id  a      b
0  1 [a1, b1] [a2, b2, c2]
1  2  [c1]    [d2, e2]
df[df.id, df.a.explode(), df.b]
  id a      b
0  1 a1 [a2, b2, c2]
1  1 b1 [a2, b2, c2]
2  2 c1  [d2, e2]
df[df.id, df.a.explode(), df.b.explode()]
  id a b
0  1 a1 a2
1  1 a1 b2
2  1 a1 c2
3  1 b1 a2
4  1 b1 b2
5  1 b1 c2
6  2 c1 d2
7  2 c1 e2
```

If the lateral view method does not produce output for a row, the row does not appear in the output by default. To display the row in the output, you can set `keep_nulls` to `True`. In this case, the column value for the row is `None` in the output.

```
df
  id  a
0  1 [a1, b1]
1  2  []
df[df.id, df.a.explode()]
  id a
0  1 a1
1  1 b1
df[df.id, df.a.explode(keep_nulls=True)]
  id a
0  1 a1
1  1 b1
2  2 None
```

For more information about how to generate a lateral view by using the `explode` method, see [Collection-related operations](#).

Limit output records

Output the first three records only.

```
iris[:3]
  sepallength  sepalwidth  petallength  petalwidth    name
0      5.1      3.5      1.4      0.2 Iris-setosa
1      4.9      3.0      1.4      0.2 Iris-setosa
2      4.7      3.2      1.3      0.2 Iris-setosa
```

For MaxCompute SQL backend, `start` and `step` methods are not supported in slice operations. But you can use the `limit` method.

```
iris.limit(3)
  sepallength  sepalwidth  petallength  petalwidth    name
0      5.1      3.5      1.4      0.2 Iris-setosa
1      4.9      3.0      1.4      0.2 Iris-setosa
2      4.7      3.2      1.3      0.2 Iris-setosa
```

 **Note** You can perform slice operations on Collection objects only, not Sequence objects.

6.6.5. Execution

This topic describes the execution methods that are supported in DataFrame API operations.

DataFrame execution method

Deferred execution

DataFrame API operations are not automatically executed. These operations are executed only when you explicitly call the `execute` method or when you call the methods that internally call `execute`. The following table lists the methods that internally call the execute method.

Method	Description	Return value
<code>persist</code>	Saves the execution results to MaxCompute tables.	PyODPS DataFrame
<code>execute</code>	Executes the operations and returns all the results.	ResultFrame
<code>head</code>	Executes the operations and returns the first N rows of result data.	ResultFrame

Method	Description	Return value
tail	Executes the operations and returns the last N rows of result data.	ResultFrame
to_pandas	Converts a Collection object to a pandas DataFrame object or a Sequence object to a Series object. If you set the wrap parameter to True, an Alibaba Cloud MaxCompute SDK for python (PyODPS) DataFrame object is returned.	- If the wrap parameter is set to True, a PyODPS DataFrame object is returned. - If the wrap parameter is set to False, a pandas DataFrame object is returned. The default value of the wrap parameter is False.
plot, hist, and boxplot	Plotting methods.	N/A

 **Note** In an interactive environment, PyODPS DataFrame automatically calls the `execute` method when it displays result data or when it calls the `repr` method. You do not need to manually call the `execute` method.

Example

```
>>> iris[iris.sepalwidth < 5][:5]
   sepalwidth  sepalwidth  petalwidth  petalwidth  name
0         4.9         3.0         1.4         0.2  Iris-setosa
1         4.7         3.2         1.3         0.2  Iris-setosa
2         4.6         3.1         1.5         0.2  Iris-setosa
3         4.6         3.4         1.4         0.3  Iris-setosa
4         4.4         2.9         1.4         0.2  Iris-setosa
```

To disable automatic execution, run the following code:

```

>>> from odps import options
>>> options.interactive = False
>>>
>>> iris[iris.sepalength < 5][:5]
Collection: ref_0
odps.Table
  name: odps_test_sqltask_finance.`pyodps_iris`
  schema:
    sepalength      : double
    sepalwidth      : double
    petallength     : double
    petalwidth      : double
    name            : string

Collection: ref_1
Filter[collection]
  collection: ref_0
  predicate:
    Less[sequence(boolean)]
      sepalength = Column[sequence(float64)] 'sepalength' from collection ref_0
      Scalar[int8]
        5

Slice[collection]
  collection: ref_1
  stop:
    Scalar[int8]
      5

```

If you call `repr` to return a printable representation of the object, the entire abstract syntax tree is displayed.

 **Note** The ResultFrame is a result set and cannot be used in subsequent calculations.

To retrieve all records from the ResultFrame, use an iterative method.

```
>>> result = iris.head(3)
>>> for r in result:
>>>     print(list(r))
[5.0999999999999996, 3.5, 1.3999999999999999, 0.20000000000000001, u'Iris-setosa']
[4.9000000000000004, 3.0, 1.3999999999999999, 0.20000000000000001, u'Iris-setosa']
[4.7000000000000002, 3.2000000000000002, 1.3, 0.20000000000000001, u'Iris-setosa']
```

If pandas is installed, a ResultFrame can be converted into a pandas DataFrame or a PyODPS DataFrame that uses the pandas backend.

```
pd_df = iris.head(3).to_pandas() # Return a pandas DataFrame.
wrapped_df = iris.head(3).to_pandas(wrap=True) # Return a PyODPS DataFrame that uses the pandas backend.
```

Save results to MaxCompute tables

You can call the `persist` method to return a new DataFrame object for a Collection object. The `persist` method uses the table name as the parameter.

```
>>> iris2 = iris[iris.sepalwidth < 2.5].persist('pyodps_iris2')
>>> iris2.head(5)
```

	sepalwidth	sepalwidth	petalwidth	petalwidth	name
0	4.5	2.3	1.3	0.3	Iris-setosa
1	5.5	2.3	4.0	1.3	Iris-versicolor
2	4.9	2.4	3.3	1.0	Iris-versicolor
3	5.0	2.0	3.5	1.0	Iris-versicolor
4	6.0	2.2	4.0	1.0	Iris-versicolor

You can pass the `partitions` parameter to the `persist` method to create a partitioned table. The table is partitioned based on the columns that are specified by `partitions`.

```
iris3 = iris[iris.sepalwidth < 2.5].persist('pyodps_iris3', partitions=['name'])
iris3.data
odps.Table
name: odps_test_sqltask_finance.`pyodps_iris3`
schema:
  sepallength      : double
  sepalwidth       : double
  petallength      : double
  petalwidth       : double
partitions:
  name             : string
```

To write data to a partition of an existing table, you can pass the `partition` parameter to the `persist` method. The partition parameter specifies the destination partition. For example, set the partition parameter to `ds=*****`. The table must contain all columns of the DataFrame object and the columns must be of the same type. The `drop_partition` and `create_partition` parameters are valid only if the partition parameter is specified. The `drop_partition` parameter specifies whether to delete the specified partition if the partition exists. The `create_partition` parameter specifies whether to create the specified partition if the partition does not exist.

```
iris[iris.sepalwidth < 2.5].persist('pyodps_iris4', partition='ds=test', drop_partition=True, create_partition=True)
```

When you write data to a table, you can specify the time-to-live (TTL) of the table. For example, the following statement sets the TTL of the table to 10 days.

```
iris[iris.sepalwidth < 2.5].persist('pyodps_iris5', lifecycle=10)
```

If the data source does not contain MaxCompute objects, you must manually specify the MaxCompute object or mark the entrance object as global when you call the `persist` method.

```
# Assume that the entrance object of PyODPS is o.
# Specify the entrance object.
df.persist('table_name', odps=o)
# Alternatively, mark the entrance object as global.
o.to_global()
df.persist('table_name')
```

Save results to pandas DataFrame

You can call the `to_pandas` method to save results to pandas DataFrame objects. If the `wrap` parameter is set to True, a PyODPS DataFrame object is returned.

```
type(iris[iris.sepalwidth < 2.5].to_pandas())
pandas.core.frame.DataFrame
type(iris[iris.sepalwidth < 2.5].to_pandas(wrap=True))
odps.df.core.DataFrame
```

After data is read from **Tables**, PyODPS can execute the `open_reader` method and use `reader.to_pandas()` to convert Collection objects to pandas DataFrame objects.

Set runtime parameters

You can set the runtime parameters for methods such as `execute`, `persist`, and `to_pandas`. This setting is valid only for the MaxCompute SQL backend.

- Set global parameters. For more information, see [SQL](#).
- You can specify the `hints` parameter in these methods. This ensures that the specified runtime parameters are valid only for the current calculation.

```
iris[iris.sepalwidth < 5].to_pandas(hints={'odps.sql.mapper.split.size': 16})
```

Display details at runtime

To view the logview of an instance at runtime, you must modify the global configurations.

```
from odps import options
options.verbose = True

iris[iris.sepalwidth < 5].exclude('sepalwidth')[0:5].execute()
Sql compiled:
SELECT t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name`
FROM odps_test_sqltask_finance.`pyodps_iris` t1
WHERE t1.`sepalwidth` < 5
LIMIT 5
logview:
http://logview
```

	sepalwidth	petallength	petalwidth	name
0	3.0	1.4	0.2	Iris-setosa
1	3.2	1.3	0.2	Iris-setosa
2	3.1	1.5	0.2	Iris-setosa
3	3.4	1.4	0.3	Iris-setosa
4	2.9	1.4	0.2	Iris-setosa

You can specify a logging function.

```

my_logs = []
def my_logger(x):
    my_logs.append(x)

options.verbose_log = my_logger

iris[iris.sepalwidth < 5].exclude('sepalwidth')[5].execute()

```

	sepalwidth	petallength	petalwidth	name
0	3.0	1.4	0.2	Iris-setosa
1	3.2	1.3	0.2	Iris-setosa
2	3.1	1.5	0.2	Iris-setosa
3	3.4	1.4	0.3	Iris-setosa
4	2.9	1.4	0.2	Iris-setosa

```

print(my_logs)
['Sql compiled:', 'SELECT t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name` \nFROM odps_test\n_sqltask_finance.`pyodps_iris` t1 \nWHERE t1.`sepalwidth` < 5 \nLIMIT 5', 'logview:', u'http://logview']

```

Cache intermediate results

When PyODPS DataFrame calculates data, some Collection objects are used multiple times. To view the execution results of an intermediate process, you can call the `cache` method to mark a Collection object that you want to calculate first.

 **Note** The execution of the `cache` method is deferred. Automatic calculation is not immediately triggered after the `cache` method is called.

```

cached = iris[iris.sepalwidth < 3.5].cache()
df = cached['sepalwidth', 'name'].head(3)
df

```

	sepalwidth	name
0	4.9	Iris-setosa
1	4.7	Iris-setosa
2	4.6	Iris-setosa

```

cached.head(3) # The result can be immediately retrieved because the cached object is calculated.

```

	sepalwidth	name
0	4.9	Iris-setosa
1	4.7	Iris-setosa
2	4.6	Iris-setosa

Asynchronous and parallel execution

PyODPS DataFrame supports asynchronous operations. For the `execute`, `persist`, `head`, `tail`, and `to_pandas` methods, you can pass the `async` parameter to enable asynchronous execution. The `timeout` parameter specifies the time-out period. Asynchronous operations return **Future** objects.

```
from odps.df import Delay
delay = Delay() # Create the Delay object.

df = iris[iris.sepal_width < 5].cache() # Common dependency of subsequent expressions.
future1 = df.sepal_width.sum().execute(delay=delay) # Return the Future object. The execution is not started.
future2 = df.sepal_width.mean().execute(delay=delay)
future3 = df.sepal_length.max().execute(delay=delay)
delay.execute(n_parallel=3) # The execution starts with three concurrent threads.
|=====| 1 / 1 (100.00%) 21s
future1.result()
458.100000000000014
future2.result()
3.05400000000000007
```

In the preceding example, PyODPS DataFrame first executes the object of the shared dependency. Then, PyODPS DataFrame sets the degree of parallelism to 3 and executes objects from `future1` to `future3`. If `n_parallel` is set to 1, the execution time reaches 37 seconds.

You can pass the `async` parameter to `delay.execute` to specify whether to enable asynchronous execution. If asynchronous execution is enabled, you can also use the `timeout` parameter to specify the time-out period.

6.6.6. Column operations

This topic describes how to call the DataFrame API to perform column operations.

Column operations

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
lens = DataFrame(o.get_table('pyodps_ml_100k_lens'))
```

If you add a constant or execute a sin function to a sequence object, you add the constant or execute the sin function to all the elements of the sequence object.

Null-related functions

The DataFrame API provides null-related built-in functions, including `isnull`, `notnull`, and `fillna`. You can use the `isnull` function to check whether the value of a field is null. You can use the `notnull` function to check whether the value of a field is not null. You can use the `fillna` function to replace null values with specified values.

```
>>> iris.sepalength.isnull().head(5)
sepalength
0    False
1    False
2    False
3    False
4    False
```

Logic functions

The DataFrame API provides the logic functions `ifelse` and `switch`. You can use the `ifelse` function to check whether a `BOOLEAN` field is true. If the field is true, the parameter numbered as 0 is returned. If the field is false, the parameter numbered as 1 is returned.

```
>>> (iris.sepalength > 5).ifelse('gt5', 'lte5').rename('cmp5').head(5)
cmp5
0  gt5
1  lte5
2  lte5
3  lte5
4  lte5
```

You can use the `switch` function to handle multiple conditions.

```
>>> iris.sepalength.switch(4.9, 'eq4.9', 5.0, 'eq5.0', default='noeq').rename('equalness').head(5)
equalness
0    noeq
1  eq4.9
2    noeq
3    noeq
4  eq5.0
```

```
>>> from odps.df import switch
>>> switch(iris.sepalength == 4.9, 'eq4.9', iris.sepalength == 5.0, 'eq5.0', default='noeq').rename('equalness').head(5)
   equalness
0    noeq
1   eq4.9
2    noeq
3    noeq
4   eq5.0
```

In Python on MaxCompute (PyODPS) V0.7.8 and later, you can modify the values of a column in a dataset based on the specified conditions.

```
>>> iris[iris.sepalength > 5, 'cmp5'] = 'gt5'
>>> iris[iris.sepalength <= 5, 'cmp5'] = 'lte5'
>>> iris.head(5)
   cmp5
0  gt5
1  lte5
2  lte5
3  lte5
4  lte5
```

Mathematical operations

In numerical fields, a sequence object supports mathematical operations such as addition (+), subtraction (-), multiplication (*), and division (/). The log and sin functions are also supported.

```
>>> (iris.sepalength * 10).log().head(5)
   sepalength
0   3.931826
1   3.891820
2   3.850148
3   3.828641
4   3.912023
```

```
>>> fields = [iris.sepalength,
>>>             (iris.sepalength / 2).rename('sepalength divided by 2'),
>>>             (iris.sepalength ** 2).rename('sepalength squared')]
>>> iris[fields].head(5)
   sepalength  sepalength divided by 2  sepalength squared
0         5.1             2.55           26.01
1         4.9             2.45           24.01
2         4.7             2.35           22.09
3         4.6             2.30           21.16
4         5.0             2.50           25.00
```

The following table describes the supported arithmetic operations.

Arithmetic operation	Description
abs	Returns the absolute value of the given number.
sqrt	Returns the square root of the given number.
sin	N/A
sinh	N/A
cos	N/A
cosh	N/A
tan	N/A
tanh	N/A
arccos	N/A
arccosh	N/A
arcsin	N/A
arcsinh	N/A
arctan	N/A
arctanh	N/A
exp	Returns e raised to the power of the given number.
expm1	Returns e raised to the power of the given number, minus 1.

Arithmetic operation	Description
log	Returns the logarithm of the given number by using a supplied base.
log2	N/A
log10	N/A
log1p	$\log(1+x)$
radians	Converts the values in radians to degrees.
degrees	Converts the values in degrees to radians.
ceil	Returns the smallest integer that is no less than the given number.
floor	Returns the largest integer that is no greater than the given number.
trunc	Returns a number truncated to the specified decimal place.

You can compare a sequence object with another sequence or scalar objects.

```
>>> (iris.sepalength < 5).head(5)
sepalength
0    False
1     True
2     True
3     True
4    False
```

The DataFrame API does not support sequential comparison, such as `3 <= iris.sepalength <= 5`. However, you can use the `between` function to check whether the value of `iris.sepalength` is within a specified interval.

```
>>> (iris.sepalength.between(3, 5)).head(5)
sepalength
0    False
1     True
2     True
3     True
4     True
```

By default, the `between` function specifies an interval that includes endpoints. To specify an open interval, set the `inclusive` parameter to `False`.

```
>>> (iris.sepalength.between(3, 5, inclusive=False)).head(5)
sepalength
0    False
1     True
2     True
3     True
4    False
```

String-related operations

The DataFrame API provides a number of string-related operations for sequence or scalar objects.

```
>>> fields = [
>>>     iris.name.upper().rename('upper_name'),
>>>     iris.name.extract('Iris(.*)', group=1)
>>> ]
>>> iris[fields].head(5)
  upper_name  name
0  IRIS-SETOSA -setosa
1  IRIS-SETOSA -setosa
2  IRIS-SETOSA -setosa
3  IRIS-SETOSA -setosa
4  IRIS-SETOSA -setosa
```

The following table describes the string-related operations.

Operation	Description
<code>capitalize</code>	N/A
<code>contains</code>	Returns whether the given string contains a substring. The substring is a regular expression if the <code>regex</code> parameter is set to <code>True</code> . The <code>regex</code> parameter is set to <code>True</code> by default.
<code>count</code>	Returns the number of occurrences of the specified string.
<code>endswith</code>	Ends the given string with the specified string.
<code>startswith</code>	Starts the given string with the specified string.

Operation	Description
extract	Extracts a regular expression. If the group is not specified, substrings that satisfy the pattern of a regular expression are returned. If the group is specified, the specified group is returned.
find	Searches from left to right and returns the position of the first occurrence of the specified substring. If no matching substring exists, the value -1 is returned.
rfind	Searches from right to left and returns the position of the first occurrence of the specified substring. If no matching substring exists, the value -1 is returned.
replace	Replaces the substrings that satisfy the pattern of a regular expression with another substring. If you specify the n parameter, the substrings are replaced n times.
get	Returns the string at the specified position.
len	Returns the length of the given string.
ljust	Pads the given string with the character specified by <code>fillchar</code> on the right until the string reaches the length specified by <code>width</code> . The default pad character is a space.
rjust	Pads the given string with the character specified by <code>fillchar</code> on the left until the string reaches the length specified by <code>width</code> . The default pad character is a space.
lower	Converts the given string to lowercase.
upper	Converts the given string to uppercase.
lstrip	Removes spaces on the left side of the given string, including blank lines.
rstrip	Removes spaces on the right side of the given string, including blank lines.
strip	Removes spaces on both sides of the given string, including blank lines.
split	Splits the given string at the specified delimiter and returns a value of LIST<STRING> type.

Operation	Description
pad	Pads the given string with the character specified by <code>fillchar</code> at the specified position, which can be on the left side, right side, or both sides of the string. The default pad character is a space.
repeat	Repeats n times.
slice	Performs slice operations.
swapcase	Converts all the uppercase characters to lowercase and all the lowercase characters to uppercase in the given string.
title	Returns a titlecased version of the given string where words start with an uppercase character and the remaining characters are lowercase. This operation is the same as the <code>str.title</code> operation.
zfill	Pads the given string with the character 0 on the left side of the string until the string reaches the length specified by <code>width</code> .
isalnum	Returns True if all characters in the given string are alphanumeric. Otherwise, False is returned. This operation is the same as the <code>str.isalnum</code> operation.
isalpha	Returns True if all characters in the given string are alphabetic. Otherwise, False is returned. This operation is the same as the <code>str.isalpha</code> operation.
isdigit	Returns True if all the characters in the given string are digits. Otherwise, False is returned. This operation is the same as the <code>str.isdigit</code> operation.
isspace	Returns True if all the characters in the given string are spaces. Otherwise, False is returned. This operation is the same as the <code>str.isspace</code> operation.
islower	Returns True if all the cased characters in the given string are lowercase. Otherwise, False is returned. This operation is the same as the <code>str.islower</code> operation.

Operation	Description
isupper	Returns True if all the cased characters in the given string are uppercase. Otherwise, False is returned. This operation is the same as the <code>str.isupper</code> operation.
istitle	Returns True if the given string is a titlecased string. Otherwise, False is returned. This operation is the same as the <code>str.istitle</code> operation.
isnumeric	Returns True if all characters in the given string are numeric. Otherwise, False is returned. This operation is the same as the <code>str.isnumeric</code> operation.
isdecimal	Returns True if all characters in the given string are decimal characters. Otherwise, False is returned. This operation is the same as the <code>str.isdecimal</code> operation.
todict	Splits the given string at the specified delimiter into a dict and returns a value of DICT<STRING, STRING> type. The two input parameters are the project delimiter and key-value delimiter.
strptime	Converts the given string representing a time to the specified format. The time format is the same as the time format in the standard Python library. For more information about the time formats in Python, see Basic date and time types .

Time-related operations

You can call time-related built-in functions to manage sequence or scalar objects of DATETIME type.


```
>>> df = lens[[lens.unix_timestamp.astype('datetime').rename('dt')]]
>>> df[df.dt,
>>> df.dt.year.rename('year'),
>>> df.dt.month.rename('month'),
>>> df.dt.day.rename('day'),
>>> df.dt.hour.rename('hour')].head(5)
      dt year month day hour
0 1998-04-08 11:02:00 1998    4    8   11
1 1998-04-08 10:57:55 1998    4    8   10
2 1998-04-08 10:45:26 1998    4    8   10
3 1998-04-08 10:25:52 1998    4    8   10
4 1998-04-08 10:44:19 1998    4    8   10
```

The following table describes the time-related attributes.

Time-related attribute	Description
year	N/A
month	N/A
day	N/A
hour	N/A
minute	N/A
second	N/A
weekofyear	Returns a number representing the week of the year where the provided date falls. Monday is taken as the first day of a week.
weekday	Returns a number representing the day of the week where the provided date falls.
dayofweek	Returns a number representing the day of the week where the provided date falls.
strftime	Converts the given string representing a time to the specified format. The time format is the same as the time in the standard Python library. For more information about the time formats in Python, see Basic date and time types .

PyODPS also supports the addition and subtraction of time. For example, you can retrieve the date three days before the current date. You can subtract one date column from another to obtain the difference in the number of milliseconds.

```

>>> df
           a           b
0 2016-12-06 16:43:12.460001 2016-12-06 17:43:12.460018
1 2016-12-06 16:43:12.460012 2016-12-06 17:43:12.460021
2 2016-12-06 16:43:12.460015 2016-12-06 17:43:12.460022
>>> from odps.df import day
>>> df.a - day(3)
           a
0 2016-12-03 16:43:12.460001
1 2016-12-03 16:43:12.460012
2 2016-12-03 16:43:12.460015
>>> (df.b - df.a).dtype
int64
>>> (df.b - df.a).rename('a')
           a
0 3600000
1 3600000
2 3600000

```

The following table describes the supported DATETIME types.

Type	Description
year	N/A
month	N/A
day	N/A
hour	N/A
minute	N/A
second	N/A
millisecond	N/A

Collection-related operations

PyODPS supports collections of LIST and DICT types. You can use subscripts to retrieve an item from both types. You can also use the `len` method to retrieve the number of items in each collection.

In addition, collections of LIST and DICT types support the `explode` method. You can use the `explode` method to display the content of a collection. For collections of LIST type, the `explode` method returns one column by default. If you set the `pos` parameter to `True`, the `explode` method returns two columns. One of the columns indicates the serial number of each value in the array. The `explode` method is similar to the `enumerate` method in Python. For collections of DICT type, the `explode` method returns two columns. The two columns indicate the keys and values, respectively. You can pass in column names to the `explode` method as the names of the generated columns.

The following examples show how to use the `explode` method:

```
>>> df
  id  a          b
0  1 [a1, b1] {'a2': 0, 'b2': 1, 'c2': 2}
1  2  [c1]      {'d2': 3, 'e2': 4}
>>> df[df.id, df.a[0], df.b['b2']]
  id  a  b
0  1  a1  1
1  2  c1 NaN
>>> df[df.id, df.a.len(), df.b.len()]
  id  a  b
0  1  2  3
1  2  1  2
>>> df.a.explode()
  a
0  a1
1  b1
2  c1
>>> df.a.explode(pos=True)
  a_pos  a
0  0  a1
1  1  b1
2  0  c1
>>> # Specify column names.
>>> df.a.explode(['pos', 'value'], pos=True)
  pos  value
0  0  a1
1  1  b1
2  0  c1
>>> df.b.explode()
  b_key  b_value
0  a2    0
```

```

1  b2    1
2  c2    2
3  d2    3
4  e2    4

>>> # Specify column names.
>>> df.b.explode(['key', 'value'])
   key value
0  a2     0
1  b2     1
2  c2     2
3  d2     3
4  e2     4

```

You can use the `explode` method with `Collection` together. In this way, the columns generated by the `explode` method are combined with the original columns.

```

>>> df[df.id, df.a.explode()]
   id  a
0  1  a1
1  1  b1
2  2  c1

>>> df[df.id, df.a.explode(), df.b.explode()]
   id  a  b_key  b_value
0  1  a1   a2      0
1  1  a1   b2      1
2  1  a1   c2      2
3  1  b1   a2      0
4  1  b1   b2      1
5  1  b1   c2      2
6  2  c1   d2      3
7  2  c1   e2      4

```

In addition to the `len` and `explode` methods, collections of LIST type support the following two methods.

Method	Description
<code>contains(v)</code>	Checks whether the given list contains a specified element.
<code>sort</code>	Sorts the given list and returns a value of LIST type.

Collections of DICT type also support the following two methods.

Method	Description
keys	Retrieves DICT keys and returns a value of LIST type.
values	Retrieves DICT values and returns a value of LIST type.

Other operations

You can use the `isin` operation to check whether the elements of a sequence object exist in a specified collection. You can use the `notin` operation to check whether the elements of a sequence object do not exist in a specified collection.

```
>>> iris.sepalength.isin([4.9, 5.1]).rename('sepalength').head(5)
sepalength
0    True
1    True
2   False
3   False
4   False
```

You can use the `cut` operation to divide data in a sequence object into several segments.

```
>>> iris.sepalength.cut(range(6), labels=['0-1', '1-2', '2-3', '3-4', '4-5']).rename('sepalength_cut').head(5)
sepalength_cut
0      None
1      4-5
2      4-5
3      4-5
4      4-5
```

You can use the `include_under` and `include_over` operations to specify the maximum and minimum values, respectively.

```
>>> labels = ['0-1', '1-2', '2-3', '3-4', '4-5', '5-']
>>> iris.sepalength.cut(range(6), labels=labels, include_over=True).rename('sepalength_cut').head(5)

```

	sepalength_cut
0	5-
1	4-5
2	4-5
3	4-5
4	4-5

Call built-in functions or UDFs in MaxCompute

To call built-in functions or user defined functions (UDFs) in MaxCompute to create columns, you can use the `func` function. The returned value of the `func` function is of `STRING` type by default. You can use the `rtype` parameter to specify the type of the returned value.

```
>>> from odps.df import func
>>>
>>> iris[iris.name, func.rand(rtype='float').rename('rand')][:4]
>>> iris[iris.name, func.rand(10, rtype='float').rename('rand')][:4]
>>> # Call UDFs defined in MaxCompute. You must specify the column name if the column name cannot
    be automatically determined.
>>> iris[iris.name, func.your_udf(iris.sepalwidth, iris.sepalength, rtype='float').rename('new_col')]
>>> # Call UDFs from other projects. You can specify the column name by using the name parameter.
>>> iris[iris.name, func.your_udf(iris.sepalwidth, iris.sepalength, rtype='float', project='udf_project', name='new_col')]
```

 **Note** The Pandas backend does not allow you to execute expressions that contain the `func` function.

6.6.7. Sort, deduplicate, sample, and transform data

This topic describes the operations you can perform on a `DataFrame`. You can sort, deduplicate, sample, and scale data. You can also process null value.

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
```

Sort data

You can sort data only in a collection.

- Call the `sort` method or `sort_values` method to sort data.

```
>>> iris.sort('sepalwidth').head(5)
```

	sepalwidth	sepalwidth	petalwidth	petalwidth	name
0	5.0	2.0	3.5	1.0	Iris-versicolor
1	6.2	2.2	4.5	1.5	Iris-versicolor
2	6.0	2.2	5.0	1.5	Iris-virginica
3	6.0	2.2	4.0	1.0	Iris-versicolor
4	5.5	2.3	4.0	1.3	Iris-versicolor

- To sort data in descending order, set the `ascending` parameter to False.

```
>>> iris.sort('sepalwidth', ascending=False).head(5)
```

	sepalwidth	sepalwidth	petalwidth	petalwidth	name
0	5.7	4.4	1.5	0.4	Iris-setosa
1	5.5	4.2	1.4	0.2	Iris-setosa
2	5.2	4.1	1.5	0.1	Iris-setosa
3	5.8	4.0	1.2	0.2	Iris-setosa
4	5.4	3.9	1.3	0.4	Iris-setosa

- You can also set `-` to sort data in descending order.

```
>>> iris.sort(-iris.sepalwidth).head(5)
```

	sepalwidth	sepalwidth	petalwidth	petalwidth	name
0	5.7	4.4	1.5	0.4	Iris-setosa
1	5.5	4.2	1.4	0.2	Iris-setosa
2	5.2	4.1	1.5	0.1	Iris-setosa
3	5.8	4.0	1.2	0.2	Iris-setosa
4	5.4	3.9	1.3	0.4	Iris-setosa

- You can sort multiple fields simultaneously in a DataFrame.

```
>>> iris.sort(['sepalwidth', 'petalwidth']).head(5)
```

	sepalwidth	sepalwidth	petalwidth	petalwidth	name
0	5.0	2.0	3.5	1.0	Iris-versicolor
1	6.0	2.2	4.0	1.0	Iris-versicolor
2	6.2	2.2	4.5	1.5	Iris-versicolor
3	6.0	2.2	5.0	1.5	Iris-virginica
4	4.5	2.3	1.3	0.3	Iris-setosa

- If you need to sort multiple fields in different orders, add a list of BOOLEAN type values for the `ascending` parameter. The length of the list must be equal to the length of the sorted fields.

```
>>> iris.sort(['sepalwidth', 'petallength'], ascending=[True, False]).head(5)
```

	sepalwidth	petallength	petalwidth	name
0	5.0	2.0	3.5	Iris-versicolor
1	6.0	2.2	5.0	Iris-virginica
2	6.2	2.2	4.5	Iris-versicolor
3	6.0	2.2	4.0	Iris-versicolor
4	6.3	2.3	4.4	Iris-versicolor

You can also sort multiple fields in different orders in the following way:

```
>>> iris.sort(['sepalwidth', -iris.petallength]).head(5)
```

	sepalwidth	petallength	petalwidth	name
0	5.0	2.0	3.5	Iris-versicolor
1	6.0	2.2	5.0	Iris-virginica
2	6.2	2.2	4.5	Iris-versicolor
3	6.0	2.2	4.0	Iris-versicolor
4	6.3	2.3	4.4	Iris-versicolor

 **Note** MaxCompute requires that you specify the number of records you need to sort. The default value is 10000. You can use the `options.df.odps.sort.limit` parameter to modify the number of records. You can set the parameter to a value greater than 10000. However, this setting can cause an out-of-memory (OOM) error.

Deduplicate data

- To deduplicate a collection, you can call the `distinct` method in the following ways:

```
>>> iris[['name']].distinct()
```

	name
0	Iris-setosa
1	Iris-versicolor
2	Iris-virginica

```
>>> iris.distinct('name')
```

	name
0	Iris-setosa
1	Iris-versicolor
2	Iris-virginica


```
>>> iris.distinct('name', 'sepalwidth').head(3)
      name sepalwidth
0  Iris-setosa      4.3
1  Iris-setosa      4.4
2  Iris-setosa      4.5
```

- To deduplicate a sequence, you can call the `unique` method. However, the sequence that is deduplicated by using the `unique` method cannot be selected in a collection.

```
>>> iris.name.unique()
      name
0  Iris-setosa
1  Iris-versicolor
2  Iris-virginica
```

An error occurs when you use the following code:

```
>>> iris[iris.name, iris.name.unique()]
```

Sample data

To sample data from a collection, call the `sample` method. Python on MaxCompute (PyODPS) supports the following four sampling methods:

 **Note** MaxCompute DataFrame must support XFlow to execute the following sampling methods, except for sampling by parts. If MaxCompute DataFrame does not support XFlow, you can only execute the sampling methods at the backend of Pandas DataFrame.

- Sampling by parts

Data is divided into `parts` by using this sampling method. You can select the part by number.

```
>>> iris.sample(parts=10) # Split data into 10 parts and take the part numbered as 0 by default.
>>> iris.sample(parts=10, i=0) # Split data into 10 parts and take the part numbered as 0.
>>> iris.sample(parts=10, i=[2, 5]) # Split data into 10 parts and take the parts numbered as 2 and 5
.
>>> iris.sample(parts=10, columns=['name', 'sepalwidth']) # Sample the data by the values of the name and sepalwidth columns.
```

- Sampling by proportion or the number of records

You must specify the number of records or the proportion of data that you want to sample when you use this method. To enable sampling with replacement, set the `replace` parameter to `True`.

```
>>> iris.sample(n=100) # Sample 100 records.
>>> iris.sample(frac=0.3) # Sample 30% of all records.
```

- Sampling by weight

You can specify the weight column, the number of records, and the proportion of records that you want to sample when you use this method. To enable sampling with replacement, set the `replace` parameter to `True`.

```
>>> iris.sample(n=100, weights='sepal_length')
>>> iris.sample(n=100, weights='sepal_width', replace=True)
```

- Sampling by stratification

You can specify the label column that you want to stratify when you use this method. You can also specify the sampling proportion (the `frac` parameter) or the number of records (the `n` parameter) for each label. This sampling method does not support sampling with replacement.

```
>>> iris.sample(strata='category', n={'Iris Setosa': 10, 'Iris Versicolour': 10})
>>> iris.sample(strata='category', frac={'Iris Setosa': 0.5, 'Iris Versicolour': 0.4})
```

Scale data

A `DataFrame` supports data scaling based on the maximum and minimum values, average value, and standard deviation. The following data is an example for data scaling:

```
name id fid
0 name1 4 5.3
1 name2 2 3.5
2 name2 3 1.5
3 name1 4 4.2
4 name1 3 2.2
5 name1 3 4.1
```

- You can use the `min_max_scale` method to normalize data.

```
df.min_max_scale(columns=['fid'])
  name id   fid
0 name1 4 1.000000
1 name2 2 0.526316
2 name2 3 0.000000
3 name1 4 0.710526
4 name1 3 0.184211
5 name1 3 0.684211
```

- You can use the `min_max_scale` method with the `feature_range` parameter to specify the output value range. The following example shows how to keep the output values in the range of (-1, 1):

```
df.min_max_scale(columns=['fid'], feature_range=(-1, 1))
```

	name	id	fid
0	name1	4	1.000000
1	name2	2	0.052632
2	name2	3	-1.000000
3	name1	4	0.421053
4	name1	3	-0.631579
5	name1	3	0.368421

- If you need to keep the original values, you must use the `preserve` parameter. The scaled data is added in a new column. By default, the new column is named by adding the `_scaled` suffix to the original column name. You can use the `suffix` parameter to change the suffix name.

```
df.min_max_scale(columns=['fid'], preserve=True)
```

	name	id	fid	fid_scaled
0	name1	4	5.3	1.000000
1	name2	2	3.5	0.526316
2	name2	3	1.5	0.000000
3	name1	4	4.2	0.710526
4	name1	3	2.2	0.184211
5	name1	3	4.1	0.684211

- You can also use the `min_max_scale` method with the `group` parameter to specify one or more group columns and to retrieve the minimum and maximum values from the specified column to scale data.

```
df.min_max_scale(columns=['fid'], group=['name'])
```

	name	id	fid
0	name1	4	1.000000
1	name1	4	0.645161
2	name1	3	0.000000
3	name1	3	0.612903
4	name2	2	1.000000
5	name2	3	0.000000

The preceding example shows that data in both `name1` and `name2` is scaled based on the minimum and maximum values of the two groups.

- You can use the `std_scale` method to scale data based on standard normal distribution. You can also use the `std_scale` method with the `preserve` parameter to keep the original

column, and use this method with the `group` parameter to group data.

```
df.std_scale(columns=['fid'])
  name id   fid
0 name1  4  1.436467
1 name2  2  0.026118
2 name2  3 -1.540938
3 name1  4  0.574587
4 name1  3 -0.992468
5 name1  3  0.496234
```

Process null value

A DataFrame supports the feature of deleting rows with null value and the feature of filling null value. The following data is an example for processing null value:

```
id name f1 f2 f3 f4
0 0 name1 1.0 NaN 3.0 4.0
1 1 name1 2.0 NaN NaN 1.0
2 2 name1 3.0 4.0 1.0 NaN
3 3 name1 NaN 1.0 2.0 3.0
4 4 name1 1.0 NaN 3.0 4.0
5 5 name1 1.0 2.0 3.0 4.0
6 6 name1 NaN NaN NaN NaN
```

- You can use the `dropna` method to delete the rows that contain null value in the `subset` object.

```
df.dropna(subset=['f1', 'f2', 'f3', 'f4'])
  id name f1 f2 f3 f4
0 5 name1 1.0 2.0 3.0 4.0
```

- To keep the rows that contain non-null values, set the `how` parameter to all.

```
df.dropna(how='all', subset=['f1', 'f2', 'f3', 'f4'])
  id name f1 f2 f3 f4
0 0 name1 1.0 NaN 3.0 4.0
1 1 name1 2.0 NaN NaN 1.0
2 2 name1 3.0 4.0 1.0 NaN
3 3 name1 NaN 1.0 2.0 3.0
4 4 name1 1.0 NaN 3.0 4.0
5 5 name1 1.0 2.0 3.0 4.0
```

- You can use the `thresh` parameter to specify the minimum number of non-null values in a

row.

```
df.dropna(thresh=3, subset=['f1', 'f2', 'f3', 'f4'])
  id  name  f1  f2  f3  f4
0  0  name1  1.0 NaN  3.0  4.0
2  2  name1  3.0  4.0  1.0 NaN
3  3  name1  NaN  1.0  2.0  3.0
4  4  name1  1.0 NaN  3.0  4.0
5  5  name1  1.0  2.0  3.0  4.0
```

- You can call the `fillna` method to replace null value with a specified constant or values in an existing column.
 - The following example shows how to replace null value with a constant:

```
df.fillna(100, subset=['f1', 'f2', 'f3', 'f4'])
  id  name   f1   f2   f3   f4
0  0  name1   1.0 100.0   3.0   4.0
1  1  name1   2.0 100.0 100.0   1.0
2  2  name1   3.0   4.0   1.0 100.0
3  3  name1 100.0   1.0   2.0   3.0
4  4  name1   1.0 100.0   3.0   4.0
5  5  name1   1.0   2.0   3.0   4.0
6  6  name1 100.0 100.0 100.0 100.0
```

- The following example shows how to replace null value with values in an existing column:

```
df.fillna(df.f2, subset=['f1', 'f2', 'f3', 'f4'])
  id  name  f1  f2  f3  f4
0  0  name1  1.0 NaN  3.0  4.0
1  1  name1  2.0 NaN NaN  1.0
2  2  name1  3.0  4.0  1.0  4.0
3  3  name1  1.0  1.0  2.0  3.0
4  4  name1  1.0 NaN  3.0  4.0
5  5  name1  1.0  2.0  3.0  4.0
6  6  name1 NaN NaN NaN NaN
```

- A DataFrame supports backward filling and forward filling to fill null value. The following table defines the valid values of the `method` parameter.

Valid value	Definition
bfill or backfill	Backward filling
ffill or pad	Forward filling

Examples

```
df.fillna(method='bfill', subset=['f1', 'f2', 'f3', 'f4'])
  id  name  f1  f2  f3  f4
0  0  name1  1.0  3.0  3.0  4.0
1  1  name1  2.0  1.0  1.0  1.0
2  2  name1  3.0  4.0  1.0  NaN
3  3  name1  1.0  1.0  2.0  3.0
4  4  name1  1.0  3.0  3.0  4.0
5  5  name1  1.0  2.0  3.0  4.0
6  6  name1  NaN  NaN  NaN  NaN
df.fillna(method='ffill', subset=['f1', 'f2', 'f3', 'f4'])
  id  name  f1  f2  f3  f4
0  0  name1  1.0  1.0  3.0  4.0
1  1  name1  2.0  2.0  2.0  1.0
2  2  name1  3.0  4.0  1.0  1.0
3  3  name1  NaN  1.0  2.0  3.0
4  4  name1  1.0  1.0  3.0  4.0
5  5  name1  1.0  2.0  3.0  4.0
6  6  name1  NaN  NaN  NaN  NaN
```

You can use the `ffill` or `bfill` function in the preceding example to simplify the code. The `ffill` function equals `fillna(method='ffill')`, and the `bfill` function equals `fillna(method='bfill')`.

6.6.8. Use UDFs and the third-party Python libraries

This topic describes how to use user-defined functions (UDFs) and the third-party Python libraries.

Use UDFs

`DataFrame` allows you to use the `map` method for a Sequence object to call UDFs on all of its elements.

```
>>> iris.sepalength.map(lambda x: x + 1).head(5)
  sepalength
0      6.1
1      5.9
2      5.7
3      5.6
4      6.0
```

 **Note** When you use UDFs, you cannot use List or Dict types as input or output.

If the type of the Sequence is changed after the `map` method is executed, you need to explicitly specify the new type of the Sequence.

```
>>> iris.sepalength.map(lambda x: 't'+str(x), 'string').head(5)
sepalength
0      t5.1
1      t4.9
2      t4.7
3      t4.6
4      t5.0
```

If a function contains a closure, the change of a closure variable value outside the function causes the variable value within the function to change.

```
>>> dfs = []
>>> for i in range(10):
>>>     dfs.append(df.sepal_length.map(lambda x: x + i))
```

As a result, each `SequenceExpr` object in `dfs` is `df.sepal_length + 9`. To solve this problem, you can use the function as the return value of another function, or use `partial`. See the following two examples.

```
>>> dfs = []
>>> def get_mapper(i):
>>>     return lambda x: x + i
>>> for i in range(10):
>>>     dfs.append(df.sepal_length.map(get_mapper(i)))
```

```
>>> import functools
>>> dfs = []
>>> for i in range(10):
>>>     dfs.append(df.sepal_length.map(functools.partial(lambda v, x: x + v, i)))
```

The `map` method also supports existing UDFs. You can pass in str type parameters, which represent function names, or Function objects. For more information, see [Functions](#).

When you implement the `map` method for Python functions, you need to use MaxCompute Python UDF. If your project does not support Python UDFs, you cannot use the `map` method. In addition, all Python UDF limits apply.

The only available third-party library (contains the code in C language) is NumPy. For more information about how to use the third-party Python libraries, see [Use the third-party Python libraries](#).

In addition to UDFs, DataFrame provides many built-in functions, some of which are implemented by using the `map` function. Therefore, if your project does not support Python UDF, you cannot use these functions. Note that Alibaba Cloud public services do not support Python UDF.

 **Note** Because of the difference in byte code definitions, if you use the new features supported by Python 3 (such as `yield from`), errors may occur when the code is executed on MaxCompute Worker of Python 2.7. Therefore, we recommend that you make sure your code runs normally before you write production code by using MapReduce API in Python 3.

Example of counter usage:

```
from odps.udf import get_execution_context
def h(x):
    ctx = get_execution_context()
    counters = ctx.get_counters()
    counters.get_counter('df', 'add_one').increment(1)
    return x + 1
df.field.map(h)
```

You can find the counter value in JSONSummary of LogView.

Use UDFs for one row

To use UDFs for one row, you can use the `apply` method. The `axis` parameter must be set to 1 to indicate that the operation is performed on the row. The UDF of the `apply` method takes a parameter, which is a row of data from the preceding Collection object. You can retrieve the data in a field by using the attribute or offset.

- If the `reduce` parameter is set to `True`, a Sequence object is returned. Otherwise, a Collection object is returned. You can use the `names` and `types` parameters to specify the field names and types of the returned Sequence or Collection object. By default, if you do not specify a type, `STRING` type is used.

```
>>> iris.apply(lambda row: row.sepalength + row.sepalwidth, axis=1, reduce=True, types='float').re
name('sepaladd').head(3)
   sepaladd
0      8.6
1      7.9
2      7.9
```

- If `reduce` is `False` in the UDF of the `apply` method, you can use the `yield` keyword to return multiple rows of results.


```
>>> iris.count()
150
>>>
>>> def handle(row):
>>>     yield row.sepalength - row.sepalwidth, row.sepalength + row.sepalwidth
>>>     yield row.petallength - row.petalwidth, row.petallength + row.petalwidth
>>>
>>> iris.apply(handle, axis=1, names=['iris_add', 'iris_sub'], types=['float', 'float']).count()
300
```

- You can also annotate the returned field names and types in the function. You do not need to specify them when you call the the function.

```
>>> from odps.df import output
>>>
>>> @output(['iris_add', 'iris_sub'], ['float', 'float'])
>>> def handle(row):
>>>     yield row.sepalength - row.sepalwidth, row.sepalength + row.sepalwidth
>>>     yield row.petallength - row.petalwidth, row.petallength + row.petalwidth
>>>
>>> iris.apply(handle, axis=1).count()
300
```

- You can also use `map_reduce` of `map-only` , which is equivalent to the `apply` method with `axis=1` .

```
>>> iris.map_reduce(mapper=handle).count()
300
```

- To use an existing user-defined table function (UDTF) in MaxCompute, specify the name of the UDTF.

```
>>> iris['name', 'sepalength'].apply('your_func', axis=1, names=['name2', 'sepalength2'], types=['string', 'float'])
```

- When you use the `apply` method for rows and `reduce` is `False`, you can use the lateral view with the existing rows for purposes such as aggregation.

```
>>> from odps.df import output
>>>
>>> @output(['iris_add', 'iris_sub'], ['float', 'float'])
>>> def handle(row):
>>>     yield row.sepalength - row.sepalwidth, row.sepalength + row.sepalwidth
>>>     yield row.petallength - row.petalwidth, row.petallength + row.petalwidth
>>>
>>> iris[iris.category, iris.apply(handle, axis=1)]
```

Use custom aggregations for all columns

When you use the `apply` method, if `axis` is not specified or the value of `axis` is 0, you can pass in a custom aggregation class to aggregate all Sequence objects.

```
class Agg(object):

    def buffer(self):
        return [0.0, 0]

    def __call__(self, buffer, val):
        buffer[0] += val
        buffer[1] += 1

    def merge(self, buffer, pBuffer):
        buffer[0] += pBuffer[0]
        buffer[1] += pBuffer[1]

    def getvalue(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
```

```
>>> iris.exclude('name').apply(Agg)
sepalength_aggregation sepalwidth_aggregation petallength_aggregation petalwidth_aggregation
0          5.843333          3.054          3.758667          1.198667
```

 **Note** When you use UDFs, you cannot use the LIST or DICT types as input or output.

Reference resources

UDFs can also read MaxCompute resources (such as table and file resources) or reference a Collection object as resources. To do this, you need to write the UDFs as a closure or callable class. See the following two examples.

```
>>> file_resource = o.create_resource('pyodps_iris_file', 'file', file_obj='Iris-setosa')
>>>
>>> iris_names_collection = iris.distinct('name')[2]
>>> iris_names_collection
      sepallength
0  Iris-setosa
1  Iris-versicolor
```

```
>>> def myfunc(resources): # resources are passed in by calling order
>>>     names = set()
>>>     fileobj = resources[0] # file resources are represented by a file-like object
>>>     for l in fileobj:
>>>         names.add(l)
>>>     collection = resources[1]
>>>     for r in collection:
>>>         names.add(r.name) # retrieve the values by using the field name or offset
>>>     def h(x):
>>>         if x in names:
>>>             return True
>>>         else:
>>>             return False
>>>     return h
>>>
>>> df = iris.distinct('name')
>>> df = df[df.name,
>>>         df.name.map(myfunc, resources=[file_resource, iris_names_collection], rtype='boolean').rename('isin')]
>>>
>>> df
      name  isin
0  Iris-setosa  True
1  Iris-versicolor  True
2  Iris-virginica  False
```

 **Note** When the partitioned tables are read, partition fields are not included.

If `axis` is 1 in a row operation, you need to write a function closure or callable class. For aggregation operations on columns, you only need to use the `__init__` function to read resources.

```
>>> words_df
      sentence
0      Hello World
1      Hello Python
2 Life is short I use Python
>>>
>>> import pandas as pd
>>> stop_words = DataFrame(pd.DataFrame({'stops': ['is', 'a', 'I']}))
>>>
>>> @output(['sentence'], ['string'])
>>> def filter_stops(resources):
>>>     stop_words = set([r[0] for r in resources[0]])
>>>     def h(row):
>>>         return ' '.join(w for w in row[0].split() if w not in stop_words),
>>>     return h
>>>
>>> words_df.apply(filter_stops, axis=1, resources=[stop_words])
      sentence
0      Hello World
1      Hello Python
2 Life short use Python
```

 **Note** In this example, `stop_words` is a local variable, which is referenced as a resource in MaxCompute during execution.

Use the third-party Python libraries

You can upload the third-party Python packages as resources to MaxCompute. Supported formats include *whl*, *egg*, *zip*, and *tar.gz*. When you use a global method or an immediately-invoked method, you need to specify the packages you need before you can use the third-party libraries in UDFs. You also need to specify the dependencies of the third-party libraries. Otherwise, errors may occur when you import the files.

- You can upload resources by calling the PyODPS resource upload API operation.

Take a *python-dateutil* package as an example.

- i. You can run the `pip download` command to download the package and its dependencies to a specific path. Two packages are downloaded: *six-1.10.0-py2.py3-none-any.whl* and *python_dateutil-2.5.3-py2.py3-none-any.whl*. Note that the downloaded packages must support the Linux environment.

```
$ pip download python-dateutil -d /to/path/
```

ii. Then upload the two files to MaxCompute as resources.

```
>>> # make sure that file extensions are correct.
>>> odps.create_resource('six.whl', 'file', file_obj=open('six-1.10.0-py2.py3-none-any.whl', 'rb'))
>>> odps.create_resource('python_dateutil.whl', 'file', file_obj=open('python_dateutil-2.5.3-py2.py3-none-any.whl', 'rb'))
```

iii. Now you have a DataFrame object that only contains a STRING field.

```
>>> df
      datestr
0  2016-08-26 14:03:29
1  2015-08-26 14:03:29
```

iv. Use the following third-party libraries in global configurations.

```
>>> from odps import options
>>>
>>> def get_year(t):
>>>     from dateutil.parser import parse
>>>     return parse(t).strftime('%Y')
>>>
>>> options.df.libraries = ['six.whl', 'python_dateutil.whl']
>>> df.datestr.map(get_year)
      datestr
0    2016
1    2015
```

Alternatively, use the `libraries` parameter of a method to specify the third-party libraries.

```
>>> def get_year(t):
>>>     from dateutil.parser import parse
>>>     return parse(t).strftime('%Y')
>>>
>>> df.datestr.map(get_year).execute(libraries=['six.whl', 'python_dateutil.whl'])
      datestr
0    2016
1    2015
```

By default, PyODPS supports the third-party libraries that contain pure Python code but no file operations. In later versions of MaxCompute, PyODPS also supports Python libraries that contain binary code or file operations. These libraries must be suffixed with `cp27-cp27m-manylinux1_x86_64` and uploaded as archives. The `.whl` packages must be renamed to `.zip` files. You also need to set `odps.isolation.session.enable` to `True` or enable `isolation` in your project. The following example shows how to upload and use the special functions in `scipy`.

```
>>> # packages containing binary code must be uploaded by using the archive method, and the file
    extension .whl must be replaced with .zip.
>>> odps.create_resource('scipy.zip', 'archive', file_obj=open('scipy-0.19.0-cp27-cp27m-manylinux1_x
    86_64.whl', 'rb'))
>>>
>>> # if isolation is enabled for your project, the following option is optional.
>>> options.sql.settings = { 'odps.isolation.session.enable': True }
>>>
>>> def psi(value):
>>>     # we recommend that you import the third-party libraries inside the function to avoid the err
    ors caused by the varying structure of a binary package between different operating systems.
>>>     from scipy.special import psi
>>>     return float(psi(value))
>>>
>>> df.float_col.map(psi).execute(libraries=['scipy.zip'])
```

Binary packages that only contain source code can be packed into Wheel files and uploaded in Linux Shell. The Wheel files generated in Mac and Windows cannot be used in MaxCompute.

```
python setup.py bdist_wheel
```

- You can also use MaxCompute Console to upload resources.
 - i. Most of the Python packages are supplied with `.whl` packages, including the packages containing binary files on various platforms. Therefore, you need to first find the packages that can run on MaxCompute.
 - ii. In addition, if you want to use a third-party package in MaxCompute, all dependencies must be included, which is complex. The following table lists the dependencies of the packages.

Package name	Dependencies
pandas	numpy, python-dateutil, pytz, six
scipy	numpy
scikit-learn	numpy, scipy

 **Note** The numpy package is already provided. You only need to upload python-dateutil, pytz, pandas, SciPy, sklearn, and six packages to run the pandas, scipy, and scikit-learn packages.

iii. You can find [python-dateutil-2.6.0.zip](#) in [python-dateutils](#) and download it.

□

iv. Rename the downloaded file to python-dateutil.zip, and upload the file as a resource in the MaxCompute console.

```
add archive python-dateutil.zip;
```

 **Note** Find, download, and upload [pytz-2017.2.zip](#) and [six-1.11.0.tar.gz](#) files in the same way as you do for the python-dateutil file.

v. To ensure that the packages that contain code in C language, such as Pandas, run on MaxCompute normally, you need to find the .whl package whose name contains cp27-cp27m-manylinux1_x86_64. Therefore, you need to find and download [pandas-0.20.2-cp27-cp27m-manylinux1_x86_64.whl](#), change its extension to .zip, and run `add archive pandas.zip;` on MaxCompute Console to upload it. Upload scipy and scikit-learn packages to MaxCompute in the same way as you do in the preceding procedure.

The following table lists the resources to be downloaded for all packages.

Package name	File name	Name of resource for upload
python-dateutil	python-dateutil-2.6.0.zip	python-dateutil.zip
pytz	pytz-2017.2.zip	pytz.zip
six	six-1.11.0.tar.gz	six.tar.gz
pandas	pandas-0.20.2-cp27-cp27m-manylinux1_x86_64.zip	pandas.zip
scipy	scipy-0.19.0-cp27-cp27m-manylinux1_x86_64.zip	scipy.zip
scikit-learn	scikit_learn-0.18.1-cp27-cp27m-manylinux1_x86_64.zip	sklearn.zip

Specify the third-party Python libraries

- Globally specify the libraries to be used.

```
>>> from odps import options
>>> options.df.libraries = ['six.whl', 'python_dateutil.whl']
```

- In an immediately-invoked method, locally specify the libraries to be used.

```
>>> df.apply(my_func, axis=1).to_pandas(libraries=['six.whl', 'python_dateutil.whl'])
```

6.6.9. MapReduce API

This topic describes the MapReduce API.

PyODPS DataFrame supports the MapReduce API. You can separately write the `map` and `reduce` functions, because a `map_reduce` process can contain only `mappers` or `reducers`.

The following example shows how to execute the `WordCount` program:

```
>>> #encoding=utf-8
>>> from odps import ODPS
>>> from odps import options
>>> options.verbose = True
>>> o = ODPS('your-access-id', 'your-secret-access-key', project='DMP_UC_dev', endpoint='http://service-corp.odps.aliyun-inc.com/api')
>>> from odps.df import DataFrame

>>> def mapper(row):
>>>     for word in row[0].split():
>>>         yield word.lower(), 1
>>>
>>> def reducer(keys):
>>>     # A list rather than cnt=0 is used. If you use cnt=0, cnt in the h function is considered a local variable, whose value is not included in the output.
>>>     cnt = [0]
>>>     def h(row, done): # done indicates that the rows with this key are iterated.
>>>         cnt[0] += row[1]
>>>         if done:
>>>             yield keys[0], cnt[0]
>>>     return h
>>> # The zx_word_count table has only one column, which is of the STRING type.
>>> word_count = DataFrame(o.get_table('zx_word_count'))
>>> table = word_count.map_reduce(mapper, reducer, group=['word', ],
>>>                               mapper_output_names=['word', 'cnt'],
>>>                               mapper_output_types=['string', 'int'],
>>>                               reducer_output_names=['word', 'cnt'],
>>>                               reducer_output_types=['string', 'int'])
>>> table
word cnt
0    are 1
1    day 1
2  doing? 1
3 everybody 1
```



```
4 first 1
5 hello 2
6 how 1
7 is 1
8 so 1
9 the 1
10 this 1
11 world 1
12 you 1
```

Use the `group` parameter to specify the field by which you want to use the `reduce` function to group data. If you do not specify this parameter, the data is grouped by all the fields. Reducers receive and initialize aggregated `keys`, and process the rows that are aggregated based on the `keys`. `done` indicates that all rows related to these `keys` are iterated.

For ease of understanding, the function is written as a closure in this example. You can also write the function as a `callable` class.

```
class reducer(object):
    def __init__(self, keys):
        self.cnt = 0

    def __call__(self, row, done): # done indicates that the rows with this key are iterated.
        self.cnt += row.cnt
        if done:
            yield row.word, self.cnt
```

The code is simplified if you use `output` for commenting.

```

>>> from odps.df import output
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def mapper(row):
>>>     for word in row[0].split():
>>>         yield word.lower(), 1
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def reducer(keys):
>>>     # A list rather than cnt=0 is used. If you use cnt=0, cnt in the h function is considered a local variable, whose value is not included in the output.
>>>     cnt = [0]
>>>     def h(row, done): # done indicates that the rows with this key are iterated.
>>>         cnt[0] += row.cnt
>>>         if done:
>>>             yield keys.word, cnt[0]
>>>     return h
>>>
>>> word_count = DataFrame(o.get_table('zx_word_count'))
>>> table = word_count.map_reduce(mapper, reducer, group='word')
    word cnt
0    are  1
1    day  1
2  doing? 1
3 everybody 1
4   first  1
5   hello  2
6    how  1
7     is  1
8    so  1
9    the  1
10   this  1
11  world  1
12   you  1

```

During iteration, you can use the `sort` parameter to sort the data by column and the `ascending` parameter to specify the sorting order. The `ascending` parameter can be a `BOOL` value, which indicates that all the fields specified by the `sort` parameter are arranged in the same order. The ascending parameter can also be a list. The number of strings in the list must be the same as the number of fields specified by the `sort` parameter.

Specify a combiner

In the MapReduce API, a `combiner` is used to aggregate data in a `mapper`. A combiner is used in the same way as a `reducer`, but a combiner cannot reference resources. The names and data types of the fields generated from a `combiner` must be the same as those from the `mapper` that corresponds to the combiner.

In this example, you can use the `reducer` as the `combiner` to aggregate data in the `mapper` to reduce shuffled data.

```
>>> words_df.map_reduce(mapper, reducer, combiner=reducer, group='word')
```

Reference resources

In the MapReduce API, you can separately specify the resources referenced by `mappers` and `reducers`.

In the following example, deprecated word filtering is performed in the `mapper`, and the number of words in a whitelist in the `reducer` is plus 5.

```

>>> white_list_file = o.create_resource('pyodps_white_list_words', 'file', file_obj='Python\nWorld')
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def mapper(resources):
>>>     stop_words = set(r[0].strip() for r in resources[0])
>>>     def h(row):
>>>         for word in row[0].split():
>>>             if word not in stop_words:
>>>                 yield word, 1
>>>     return h
>>>
>>> @output(['word', 'cnt'], ['string', 'int'])
>>> def reducer(resources):
>>>     d = dict()
>>>     d['white_list'] = set(word.strip() for word in resources[0])
>>>     d['cnt'] = 0
>>>     def inner(keys):
>>>         d['cnt'] = 0
>>>         def h(row, done):
>>>             d['cnt'] += row.cnt
>>>             if done:
>>>                 if row.word in d['white_list']:
>>>                     d['cnt'] += 5
>>>                 yield keys.word, d['cnt']
>>>         return h
>>>     return inner
>>>
>>> words_df.map_reduce(mapper, reducer, group='word',
>>>                     mapper_resources=[stop_words], reducer_resources=[white_list_file])
word cnt
0 hello 2
1 life 1
2 python 7
3 world 6
4 short 1
5 use 1

```

Use third-party Python libraries

The usage is similar to that of a third-party Python library in map stages.

- Specify the library globally:

```
>>> from odps import options
>>> options.df.libraries = ['six.whl', 'python_dateutil.whl']
```

- If you use an immediately-invoked method, specify the library locally:

```
>>> df.map_reduce(mapper=my_mapper, reducer=my_reducer, group='key').execute(libraries=['six.whl', 'python_dateutil.whl'])
```

 **Note** Due to the difference in bytecode definitions, if you write code based on new features supported by Python 3, such as `yield from`, an error is reported when the code is executed on a MaxCompute worker of Python 2.7. Before you implement production operations by using the MapReduce API of Python 3, we recommend that you make sure that the code runs normally.

Reshuffle data

If data is unevenly distributed in a cluster, you can call the `reshuffle` method to reshuffle data.

```
>>> df1 = df.reshuffle()
```

By default, data is hashed as random numbers. You can also distribute data by column and sort the reshuffled data in a specific order.

```
>>> df1.reshuffle('name', sort='id', ascending=False)
```

Bloom filter

PyODPS DataFrame provides the `bloom_filter` interface to calculate data with a Bloom filter.

If you specify a collection object and its `sequence1` for column calculation, you can apply the Bloom filter to `sequence2`. This way, the data that exists in `sequence1` rather than `sequence2` is filtered out, but the data that is not in `sequence2` may not be completely filtered out. This is an approximate method. This method quickly filters out useless data from the collection object. The Bloom filter is suitable for `JOIN` operations in the scenario where data volumes are significantly different. Most data is not `joined` in this scenario. For example, most user visits do not create transactions. If you want to `join` visit data with transaction data, you can apply the Bloom filter to visit data before you `join` visit data with transaction data. This enhances operation performance.

```
>>> df1 = DataFrame(pd.DataFrame({'a': ['name1', 'name2', 'name3', 'name1'], 'b': [1, 2, 3, 4]}))
>>> df1
   a b
0 name1 1
1 name2 2
2 name3 3
3 name1 4
>>> df2 = DataFrame(pd.DataFrame({'a': ['name1']}))
>>> df2
   a
0 name1
>>> df1.bloom_filter('a', df2.a) # The row 0 can be a computation expression, for example, df1.a + '1'.
   a b
0 name1 1
1 name1 4
```

Description:

- Small amounts of data are processed. Therefore, the rows that contain `name2` and `name3` in column `a` of `df1` are filtered out. However, if the system processes large amounts of data, the system may not filter out all the data that meets the specified conditions.
- In the preceding `JOIN` operation, the data that is not filtered out does not affect the data accuracy. However, data filtering greatly enhances the `JOIN` performance.
- You can specify the `capacity` and `error_rate` parameters to configure the data volume and error rate. The default values are `3000` and `0.01`.

 **Note** If you increase the value of the `capacity` parameter or decrease the value of the `error_rate` parameter, the usage of memory is increased. Therefore, set the parameters to proper values based on your requirements.

For more information about collection objects, see [DataFrame execution](#).

Pivot table

PyODPS DataFrame provides the pivot table feature. The following code shows the sample table data:

```
>>> df
   A  B   C D E
0 foo one small 1 3
1 foo one large 2 4
2 foo one large 2 5
3 foo two small 3 6
4 foo two small 3 4
5 bar one large 4 5
6 bar one small 5 3
7 bar two small 6 2
8 bar two large 7 1
```

- If you use the pivot table feature, the `rows` parameter is required to obtain the average value based on one or more fields.

```
>>> df['A', 'D', 'E'].pivot_table(rows='A')
   A D_mean E_mean
0 bar   5.5   2.75
1 foo   2.2   4.40
```

- You can specify multiple fields in `rows` to aggregate data based on the fields.

```
>>> df.pivot_table(rows=['A', 'B', 'C'])
   A  B   C D_mean E_mean
0 bar one large   4.0   5.0
1 bar one small   5.0   3.0
2 bar two large   7.0   1.0
3 bar two small   6.0   2.0
4 foo one large   2.0   4.5
5 foo one small   1.0   3.0
6 foo two small   3.0   5.0
```

- You can use the `values` parameter to specify the column that you want to calculate.

```
>>> df.pivot_table(rows=['A', 'B'], values='D')
   A  B   D_mean
0 bar one 4.500000
1 bar two 6.500000
2 foo one 1.666667
3 foo two 3.000000
```

- By default, the average value is calculated. You can use the `aggfunc` parameter to specify one or more aggregate functions.

```
>>> df.pivot_table(rows=['A', 'B'], values=['D'], aggfunc=['mean', 'count', 'sum'])
   A  B  D_mean  D_count  D_sum
0  bar one  4.500000     2    9
1  bar two  6.500000     2   13
2  foo one  1.666667     3    5
3  foo two  3.000000     2    6
```

- You can use the values of an original column as the values of a column in the new collection object.

```
>>> df.pivot_table(rows=['A', 'B'], values='D', columns='C')
   A  B  large_D_mean  small_D_mean
0  bar one         4.0          5.0
1  bar two         7.0          6.0
2  foo one         2.0          1.0
3  foo two         NaN          3.0
```

- You can use `fill_value` to replace empty values.

```
>>> df.pivot_table(rows=['A', 'B'], values='D', columns='C', fill_value=0)
   A  B  large_D_mean  small_D_mean
0  bar one          4          5
1  bar two          7          6
2  foo one          2          1
3  foo two          0          3
```

Key-value string conversion

DataFrame can extract key-value pairs into columns, and convert standard columns to key-value pairs. For more information about how to create and manage a DataFrame object, see [Create a DataFrame object](#).

- Extract key-value pairs into columns. Example:

```
>>> df
   name      kv
0 name1 k1=1,k2=3,k5=10
1 name1 k1=7.1,k7=8.2
2 name2 k2=1.2,k3=1.5
3 name2 k9=1.1,k2=1
```

Use the `extract_kv` method to extract key-value fields:


```
>>> df.extract_kv(columns=['kv'], kv_delim='=', item_delim=',')
  name kv_k1 kv_k2 kv_k3 kv_k5 kv_k7 kv_k9
0 name1  1.0  3.0  NaN  10.0  NaN  NaN
1 name1  7.0  NaN  NaN  NaN  8.2  NaN
2 name2  NaN  1.2  1.5  NaN  NaN  NaN
3 name2  NaN  1.0  NaN  NaN  NaN  1.1
```

The `columns` parameter specifies the fields that you want to extract. The `kv_delim` parameter specifies the delimiter between the keys and values. The `item_delim` parameter specifies the delimiter between key-value pairs. If you do not specify the parameters, keys and values are separated with colons (:), and key-value pairs are separated with commas (,). The name of the output field is the combination of the original field name and key value. The name and key value are connected by using an underscore (_). The default value for a missing column is `NONE`. You can use `fill_value` to fill the values for missing columns.

```
>>> df.extract_kv(columns=['kv'], kv_delim='=', fill_value=0)
  name kv_k1 kv_k2 kv_k3 kv_k5 kv_k7 kv_k9
0 name1  1.0  3.0  0.0  10.0  0.0  0.0
1 name1  7.0  0.0  0.0  0.0  8.2  0.0
2 name2  0.0  1.2  1.5  0.0  0.0  0.0
3 name2  0.0  1.0  0.0  0.0  0.0  1.1
```

- Convert multiple columns to key-value pairs. Example:

```
>>> df
  name k1 k2 k3 k5 k7 k9
0 name1 1.0 3.0 NaN 10.0 NaN NaN
1 name1 7.0 NaN NaN NaN 8.2 NaN
2 name2 NaN 1.2 1.5 NaN NaN NaN
3 name2 NaN 1.0 NaN NaN NaN 1.1
```

Use the `to_kv` method to convert data to the key-value format:

```
>>> df.to_kv(columns=['k1', 'k2', 'k3', 'k5', 'k7', 'k9'], kv_delim='=')
  name kv
0 name1 k1=1,k2=3,k5=10
1 name1 k1=7.1,k7=8.2
2 name2 k2=1.2,k3=1.5
3 name2 k9=1.1,k2=1
```

6.6.10. Debugging

Alibaba Cloud MaxCompute SDK for Python (PyODPS) DataFrame optimizes the execution process of each operation. You can use the visualization feature to display the entire computation process.

DataFrame visualization

DataFrame visualization depends on **graphviz** and the *graphviz* package for Python.

```
>>> df = iris.groupby('name').agg(id=iris.sepalwidth.sum())
>>> df = df[df.name, df.id + 3]
>>> df.visualize()
```

□

The computation process shows that PyODPS DataFrame combines the **groupby** operation and the column filtering operation.

```
>>> df = iris.groupby('name').agg(id=iris.sepalwidth.sum()).cache()
>>> df2 = df[df.name, df.id + 3]
>>> df2.visualize()
```

□

The entire execution process runs in two steps because a cache operation is performed.

View compiled results at the MaxCompute SQL backend

Call the **compile** method to view compiled results of SQL statements at the MaxCompute SQL backend.

```
>>> df = iris.groupby('name').agg(sepalwidth=iris.sepalwidth.max())
>>> df.compile()
Stage 1:

SQL compiled:

SELECT
  t1.`name`,
  MAX(t1.`sepalwidth`) AS `sepalwidth`
FROM test_pyodps_dev.`pyodps_iris` t1
GROUP BY
  t1.`name`
```

Perform local debugging by using the pandas computation backend

If you use MaxCompute tables as the sources for DataFrame objects, the system does not compile or perform some operations at the MaxCompute SQL backend. Instead, the system uses the MaxCompute Tunnel service to download data. This way, the system does not need to wait for the MaxCompute SQL backend to schedule download tasks. This allows you to download small amounts of data from MaxCompute to a local directory and use the pandas computation backend to compile and debug code. You can perform the following debugging operations:

- Select all or some rows of data from a non-partitioned table, or filter the column data, and then calculate the number of specified rows. The column filtering operation does not compute column values.
- Select all or some rows of data from all or the first several partition columns that are specified in a partitioned table, or filter the column data, and then calculate the number of the specified rows.

Assume that the `iris` DataFrame object uses a non-partitioned MaxCompute table as the source. The following operation uses the MaxCompute Tunnel service to download data:

```
>>> iris.count()
>>> iris['name', 'sepalwidth'][:10]
```

Assume that the DataFrame object uses a partitioned table that includes the `ds`, `hh`, and `mm` partition fields. The following operation uses the MaxCompute Tunnel service to download data:

```
>>> df[:10]
>>> df[df.ds == '20160808']['f0', 'f1']
>>> df[(df.ds == '20160808') & (df.hh == 3)][:10]
>>> df[(df.ds == '20160808') & (df.hh == 3) & (df.mm == 15)]
```

In this case, you can use the `to_pandas` method to download the data to a local directory for debugging.

```
>>> DEBUG = True
>>> if DEBUG:
>>> df = iris[:100].to_pandas(wrap=True)
>>> else:
>>> df = iris
```

At the end of compiling, set `DEBUG` to False to complete computation on MaxCompute.

 **Note** Some programs that pass local debugging may fail to run on MaxCompute because of the limits of the sandbox.

6.7. User experience enhancement

6.7.1. Command line

PyODPS provides an enhanced command line tool.

After you configure an account, you do not need to enter the account information again. You can perform the following steps to configure an account and call objects:

1. Import the PyODPS enhancement tool.

```
>>> from odps.inter import setup, enter, teardown
```

2. Configure your account.

```
>>> setup(**your-access_id**, **your-access-key**, **your-project**, endpoint=**your-endpoint**)
```

 **Note** If you do not specify the `room` parameter, the `default` `room` is used.

3. Call the enter method to create a room object on any Python interactive interface.

```
>>> room = enter()
>>> o = room.odps
>>> o.get_table('dual')
odps.Table
name: odps_test_sqltask_finance.`dual`
schema:
  c_int_a      : bigint
  c_int_b      : bigint
  c_double_a   : double
  c_double_b   : double
  c_string_a   : string
  c_string_b   : string
  c_bool_a     : boolean
  c_bool_b     : boolean
  c_datetime_a : datetime
  c_datetime_b : datetime
```

 **Note** The MaxCompute object is not automatically updated when you change the setup of the `room`. You must call the `enter` method again to retrieve the new `room` object.

After you configure an account and call objects, you can store, retrieve, or delete objects in the `room` or delete the entire `room`.

- You can store commonly used MaxCompute tables or resources in the `room`.

```
>>> room.store('stored-table', o.get_table('dual'), desc='Simple stored table example')
```

You can call the `display` method to display the stored objects as a table.

```
>>> room.display()
default name      desc
stored-table     Simple stored table example
iris             Iris dataset
```

- You can run the `room['stored-table']` or `room.iris` command to retrieve the stored objects.

```
>>> room['stored-table']
odps.Table
name: odps_test_sqltask_finance.`dual`
schema:
  c_int_a      : bigint
  c_int_b      : bigint
  c_double_a   : double
  c_double_b   : double
  c_string_a   : string
  c_string_b   : string
  c_bool_a     : boolean
  c_bool_b     : boolean
  c_datetime_a : datetime
  c_datetime_b : datetime
```

- You can call the `drop` method to delete objects from the `room`.

```
>>> room.drop('stored-table')
>>> room.display()
default name      desc
iris             Iris dataset
```

- You can call the `teardown` method to delete a `room`. If no parameters are specified, the default `room` is deleted.

```
teardown()
```

6.7.2. IPython

PyODPS provides an IPython plug-in to simplify the operations on MaxCompute.

IPython enhancement

Some commands are provided for command line enhancement.

- Run the following commands to load the plug-in:

```
%load_ext odps
%enter
```

The following result is returned:

```
<odps.inter.Room at 0x11341df10>
```

- In this case, the `o` and `odps` global variables can be retrieved. You can run the `o.get_table` or `odps.get_table` command to call a table.

```
o.get_table('dual')
odps.get_table('dual')
```

The following result is returned:

```
odps.Table
name: odps_test_sqltask_finance.`dual`
schema:
  c_int_a      : bigint
  c_int_b      : bigint
  c_double_a   : double
  c_double_b   : double
  c_string_a   : string
  c_string_b   : string
  c_bool_a     : boolean
  c_bool_b     : boolean
  c_datetime_a : datetime
  c_datetime_b : datetime
```

- Run the following command to display the stored objects as a table:

```
%stores
```

default name	desc
iris	Iris dataset

Object name completion

PyODPS enhances the code completion feature that is provided by IPython. When you write a statement such as `o.get_xxx`, the object name is automatically completed. In the following examples, `<tab>` is used to denote pressing the Tab key. When you enter the statement and encounter `<tab>`, press the Tab key.

- You can use the Tab key to complete the object name.

```
o.get_table(<tab>
```

- You can enter the first few characters of an object name and press the Tab key to complete it.

```
o.get_table('tabl<tab>
```

IPython auto-completes the table name that starts with `tabl`.

- This feature also completes the names of objects in different projects. Syntax:

```
o.get_table(project='project_name', name='tabl<tab>
o.get_table('tabl<tab>', project='project_name')
```

- If multiple matching objects exist, IPython provides a list. `options.completion_size` specifies the maximum number of objects in the list. The default value is 10.

SQL statements

PyODPS provides an SQL plug-in to execute MaxCompute SQL statements.

- You can use `%sql` to execute a single-line SQL statement.

```
In [37]: %sql select * from pyodps_iris limit 5
|=====| 1 / 1 (100.00%)    3s
Out[37]:
  sepallength  sepalwidth  petallength  petalwidth    name
0         5.1         3.5         1.4         0.2 Iris-setosa
1         4.9         3.0         1.4         0.2 Iris-setosa
2         4.7         3.2         1.3         0.2 Iris-setosa
3         4.6         3.1         1.5         0.2 Iris-setosa
4         5.0         3.6         1.4         0.2 Iris-setosa
```

- You can use `%%sql` to execute a multiple-line SQL statement.

```
In [38]: %%sql
.....: select * from pyodps_iris
.....: where sepallength < 5
.....: limit 5
.....:
|=====| 1 / 1 (100.00%)    15s
Out[38]:
  sepallength  sepalwidth  petallength  petalwidth    name
0         4.9         3.0         1.4         0.2 Iris-setosa
1         4.7         3.2         1.3         0.2 Iris-setosa
2         4.6         3.1         1.5         0.2 Iris-setosa
3         4.6         3.4         1.4         0.3 Iris-setosa
4         4.4         2.9         1.4         0.2 Iris-setosa
```

- To execute parameterized SQL statements, you can use `:parameter` to specify the parameter.

```
In [1]: %load_ext odps

In [2]: mytable = 'dual'

In [3]: %sql select * from :mytable
|=====| 1 / 1 (100.00%)    2s
Out[3]:
  c_int_a  c_int_b  c_double_a  c_double_b  c_string_a  c_string_b  c_bool_a \
0      0      0    -1203      0      0    -1203    True

  c_bool_b      c_datetime_a      c_datetime_b
0  False  2012-03-30 23:59:58  2012-03-30 23:59:59
```

- For SQL runtime parameters, you can use `%set` to set a global parameter or use `SET` within an SQLCell to set a local parameter. The following example sets a local parameter, which does not affect the global settings.

```
In [17]: %%sql
        set odps.sql.mapper.split.size = 16;
        select * from pyodps_iris;
```

The following example sets a global parameter. Global settings are applied to all subsequent SQL statements.

```
In [18]: %set odps.sql.mapper.split.size = 16
```

Upload pandas DataFrame objects to MaxCompute tables

PyODPS provides the commands to upload pandas DataFrame objects to MaxCompute tables. You only need to run the `%persist` command. The first parameter `df` is the variable name. The second parameter `pyodps_pandas_df` is the MaxCompute table name.

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.arange(9).reshape(3, 3), columns=list('abc'))
%persist df pyodps_pandas_df
```

6.7.3. Jupyter Notebook

PyODPS enhances the result exploration and progress display features of Jupyter Notebook.

Result exploration

PyODPS provides a data exploration feature in Jupyter Notebook for SQLCell and DataFrame. You can use interactive data exploration tools to browse local data and create graphs.

1. If the execution result is a DataFrame object, PyODPS reads the result and displays it in a paged table. You can click a page number, the Previous button, or the Next button to browse the data.
2. You can select other display modes on the top of the table to display the result in a column chart, pie chart, line chart, or scatter chart. The following figure shows a scatter chart created based on the default fields, which are the first three fields.
3. You can click the Settings icon in the upper-right corner of a graph to modify the settings. For example, set Groups to name, X Axis to petallength, and Y Axis to petalwidth. The result is shown in the following graph. The petallength-petalwidth settings display the data in a manner that is easy to understand.

For column charts and pie charts, you can select an aggregate function for the value fields. The default aggregate function for column charts is `sum`, and that for pie charts is `count`. You can click the function name next to the name of the value field to select another function.

For line charts, the values on the x-axis cannot be null. If any values are null, the graph may not be correctly displayed.

4. After you make the graph, click the Download icon to save it.

 **Note** To use this feature, you must install pandas and ipywidgets.

Progress display

The execution of large jobs takes extended periods of time. PyODPS provides progress bars to show the execution progress. When DataFrame jobs, machine learning jobs, or SQL statements that start with `%sql` are executed in Jupyter Notebook, a list of these jobs and their overall progress are displayed.

If you click a job name, a dialog box that shows the progress of each task in the job appears.

After the execution is completed, a message that indicates whether the job succeeded appears.

6.8. API overview

This topic provides the links to PyODPS API documentation, which provides parameter descriptions and examples of each function:

- [Definitions](#)
- [DataFrame Reference](#)

6.9. Examples

6.9.1. Reference a third-party package in a PyODPS node

This topic describes how to reference a third-party package in a PyODPS node.

Prerequisites

The following operations are completed:

- [Activate MaxCompute](#)
- [Activate DataWorks](#)

Procedure

1. Download the files that are listed in the following table.

Package name	File name	Resource name
python-dateutil	python-dateutil-2.6.0.zip	python-dateutil.zip
pytz	pytz-2017.2.zip	pytz.zip
six	six-1.11.0.tar.gz	six.tar.gz
pandas	pandas-0.20.2-cp27-cp27m-manylinux1_x86_64.whl	pandas.zip
scipy	scipy-0.19.0-cp27-cp27m-manylinux1_x86_64.whl	scipy.zip
scikit-learn	scikit_learn-0.18.1-cp27-cp27m-manylinux1_x86_64.whl	sklearn.zip

2. Log on to the [DataWorks console](#).
3. Create a workflow.
 - i. On the homepage of the DataWorks console, click Workspaces in the left-side navigation pane. On the Workspaces page, find the target workspace and click Data Analytics in the Actions column. On the Data Development tab, right-click Business process and select New business process.
 - ii. In the New business process dialog box, set the Business Name parameter and click New.
4. Create and commit resources.
 - i. Right-click the workflow that you created in the Business process section and choose New > MaxCompute > Resource > Archive.
 - ii. In the New resource dialog box, click Click Upload and upload the *python-dateutil-2.6.0.zip* package.

- iii. Set the Resource Name parameter to *python.dateutil.zip* and click Confirm.

- iv. Click the Submit icon in the toolbar.

- v. Use the same method to create and commit the resources named *pytz.zip*, *six.tar.gz*, *pandas.zip*, *sklearn.zip*, and *scipy.zip*.

5. Create a PyODPS 2 node.

- i. Right-click the workflow that you created and choose **New > MaxCompute > PyODPS 2**.
- ii. In the New node dialog box, set the **Node name** parameter and click **Submit**. We recommend that you set the Node name parameter to a value that describes the use of third-party packages.
- iii. On the configuration tab of the PyODPS 2 node, enter the code of the node in the code editor. In this example, enter the following code:

```
def test(x):  
    from sklearn import datasets, svm  
    from scipy import misc  
    import numpy as np  
  
    iris = datasets.load_iris()  
    assert iris.data.shape == (150, 4)  
    assert np.array_equal(np.unique(iris.target), [0, 1, 2])  
  
    clf = svm.LinearSVC()  
    clf.fit(iris.data, iris.target)  
    pred = clf.predict([[5.0, 3.6, 1.3, 0.25]])  
    assert pred[0] == 0  
  
    assert misc.face().shape is not None  
  
    return x  
  
from odps import options  
  
hints = {  
    'odps.isolation.session.enable': True  
}  
libraries = ['python-dateutil.zip', 'pytz.zip', 'six.tar.gz', 'pandas.zip', 'scipy.zip', 'sklearn.zip']  
  
iris = o.get_table('pyodps_iris').to_df()  
  
print iris[:1].sepallength.map(test).execute(hints=hints, libraries=libraries)
```

6. Click the Run icon in the toolbar.
7. View the running result of the PyODPS 2 node on the Run Log tab.

Sql compiled:

```
CREATE TABLE tmp_pyodps_a3172c30_a0d7_4c88_bc39_434168263897 LIFECYCLE 1 AS
SELECT pyodps_udf_1576485276_94d9d978_af66_4e27_a874_e787022dfb3d(t1.`sepallength`) AS `sepallength`
FROM WB_BestPractice_dev.`pyodps_iris` t1
LIMIT 1
```

Instance ID: 20191216083438175gcv6n4pr2

Log view: <http://logview.odps.aliyun.com/logview/?h=xxxxxx>

sepallength

0 5.1

 **Note** For the best practices, see [Use a PyODPS node to segment Chinese text based on jieba](#).

6.9.2. Use a PyODPS node to query data based on specific criteria

This topic describes how to use a PyODPS node to query data based on specific criteria.

Prerequisites

- [MaxCompute is activated](#).
- [DataWorks is activated](#).
- A workflow is created in the DataWorks console. In this example, a workflow is created for a DataWorks workspace in basic mode. For more information, see [Create a workflow](#).

Procedure

1. Create a table and import data to it.
 - i. Download the [Iris](#) dataset named *iris.data* and rename it *iris.csv*.

- ii. Create a table named `pyodps_iris` and upload the *iris.csv* dataset. For more information, see [Create tables and import data](#). In this example, execute the following statement to create the `pyodps_iris` table:

```
CREATE TABLE if not exists pyodps_iris
(
  sepallength DOUBLE comment 'sepal length (cm)',
  sepalwidth  DOUBLE comment 'sepal width (cm)',
  petallength DOUBLE comment 'petal length (cm)',
  petalwidth  DOUBLE comment 'petal width (cm)',
  name        STRING comment 'type'
);
```

2. Log on to the [DataWorks console](#). Click Workspaces in the left-side navigation pane. On the Workspaces page, find the required workspace and click Data Analytics in the Actions column. The Data Development tab appears.
3. In the left-side navigation pane, click **Workspaces**.
4. On the page that appears, find the workspace that you want to manage and click **Data Analytics** in the Actions column.
5. On the DataStudio page, right-click a workflow and choose **New > MaxCompute > PyODPS 2**.
6. In the **New node** dialog box, specify the **Node name** parameter and click **Submit**.
7. On the configuration tab, enter sample code in the code editor.

```
import sys
reload(sys)
# Specify the encoding format.
sys.setdefaultencoding('utf8')

iris = DataFrame(o.get_table('pyodps_iris'))
# Method 1: Return the result based on a conditional query statement.
with o.execute_sql('select * from pyodps_iris WHERE sepallength > 5 ').open_reader() as reader4:
    print reader4.raw
    for record in reader4:
        print record["sepallength"],record["sepalwidth"],record["petallength"],record["petalwidth"],record["name"]

# Method 2: Return the result based on the filter conditions of DataFrame.
print iris[iris.sepallength > 5].head(5)

# Method 3: Return the result by using the query method of PyODPS DataFrame.
print iris.query("(sepallength < 5) and (petallength > 1.5)").head(5)
```

8. Click the icon.

9. View the running result of the PyODPS 2 node on the Run Log tab.

Executing user script with PyODPS 0.8.0

```
"sepallength","sepalwidth","petallength","petalwidth","name"
5.1,3.5,1.4,0.2,"Iris-setosa"
5.4,3.9,1.7,0.4,"Iris-setosa"
5.4,3.7,1.5,0.2,"Iris-setosa"
5.8,4.0,1.2,0.2,"Iris-setosa"
5.7,4.4,1.5,0.4,"Iris-setosa"
5.4,3.9,1.3,0.4,"Iris-setosa"
5.1,3.5,1.4,0.3,"Iris-setosa"
5.7,3.8,1.7,0.3,"Iris-setosa"
5.1,3.8,1.5,0.3,"Iris-setosa"
5.4,3.4,1.7,0.2,"Iris-setosa"
5.1,3.7,1.5,0.4,"Iris-setosa"
5.1,3.3,1.7,0.5,"Iris-setosa"
5.2,3.5,1.5,0.2,"Iris-setosa"
5.2,3.4,1.4,0.2,"Iris-setosa"
5.4,3.4,1.5,0.4,"Iris-setosa"
5.2,4.1,1.5,0.1,"Iris-setosa"
5.5,4.2,1.4,0.2,"Iris-setosa"
5.5,3.5,1.3,0.2,"Iris-setosa"
5.1,3.4,1.5,0.2,"Iris-setosa"
5.1,3.8,1.9,0.4,"Iris-setosa"
5.1,3.8,1.6,0.2,"Iris-setosa"
5.3,3.7,1.5,0.2,"Iris-setosa"
7.0,3.2,4.7,1.4,"Iris-versicolor"
6.4,3.2,4.5,1.5,"Iris-versicolor"
6.9,3.1,4.9,1.5,"Iris-versicolor"
5.5,2.3,4.0,1.3,"Iris-versicolor"
6.5,2.8,4.6,1.5,"Iris-versicolor"
5.7,2.8,4.5,1.3,"Iris-versicolor"
6.3,3.3,4.7,1.6,"Iris-versicolor"
6.6,2.9,4.6,1.3,"Iris-versicolor"
5.2,2.7,3.9,1.4,"Iris-versicolor"
5.9,3.0,4.2,1.5,"Iris-versicolor"
6.0,2.2,4.0,1.0,"Iris-versicolor"
6.1,2.9,4.7,1.4,"Iris-versicolor"
5.6,2.9,3.6,1.3,"Iris-versicolor"
```

```
6.7,3.1,4.4,1.4,"Iris-versicolor"  
5.6,3.0,4.5,1.5,"Iris-versicolor"  
5.8,2.7,4.1,1.0,"Iris-versicolor"  
6.2,2.2,4.5,1.5,"Iris-versicolor"  
5.6,2.5,3.9,1.1,"Iris-versicolor"  
5.9,3.2,4.8,1.8,"Iris-versicolor"  
6.1,2.8,4.0,1.3,"Iris-versicolor"  
6.3,2.5,4.9,1.5,"Iris-versicolor"  
6.1,2.8,4.7,1.2,"Iris-versicolor"  
6.4,2.9,4.3,1.3,"Iris-versicolor"  
6.6,3.0,4.4,1.4,"Iris-versicolor"  
6.8,2.8,4.8,1.4,"Iris-versicolor"  
6.7,3.0,5.0,1.7,"Iris-versicolor"  
6.0,2.9,4.5,1.5,"Iris-versicolor"  
5.7,2.6,3.5,1.0,"Iris-versicolor"  
5.5,2.4,3.8,1.1,"Iris-versicolor"  
5.5,2.4,3.7,1.0,"Iris-versicolor"  
5.8,2.7,3.9,1.2,"Iris-versicolor"  
6.0,2.7,5.1,1.6,"Iris-versicolor"  
5.4,3.0,4.5,1.5,"Iris-versicolor"  
6.0,3.4,4.5,1.6,"Iris-versicolor"  
6.7,3.1,4.7,1.5,"Iris-versicolor"  
6.3,2.3,4.4,1.3,"Iris-versicolor"  
5.6,3.0,4.1,1.3,"Iris-versicolor"  
5.5,2.5,4.0,1.3,"Iris-versicolor"  
5.5,2.6,4.4,1.2,"Iris-versicolor"  
6.1,3.0,4.6,1.4,"Iris-versicolor"  
5.8,2.6,4.0,1.2,"Iris-versicolor"  
5.6,2.7,4.2,1.3,"Iris-versicolor"  
5.7,3.0,4.2,1.2,"Iris-versicolor"  
5.7,2.9,4.2,1.3,"Iris-versicolor"  
6.2,2.9,4.3,1.3,"Iris-versicolor"  
5.1,2.5,3.0,1.1,"Iris-versicolor"  
5.7,2.8,4.1,1.3,"Iris-versicolor"  
6.3,3.3,6.0,2.5,"Iris-virginica"  
5.8,2.7,5.1,1.9,"Iris-virginica"  
7.1,3.0,5.9,2.1,"Iris-virginica"  
6.3,2.9,5.6,1.8,"Iris-virginica"  
6.5,3.0,5.8,2.2,"Iris-virginica"  
7.6,3.0,6.6,2.1,"Iris-virginica"  
7.3,2.9,6.3,1.8,"Iris-virginica"
```



```
6.7,2.5,5.8,1.8,"Iris-virginica"  
7.2,3.6,6.1,2.5,"Iris-virginica"  
6.5,3.2,5.1,2.0,"Iris-virginica"  
6.4,2.7,5.3,1.9,"Iris-virginica"  
6.8,3.0,5.5,2.1,"Iris-virginica"  
5.7,2.5,5.0,2.0,"Iris-virginica"  
5.8,2.8,5.1,2.4,"Iris-virginica"  
6.4,3.2,5.3,2.3,"Iris-virginica"  
6.5,3.0,5.5,1.8,"Iris-virginica"  
7.7,3.8,6.7,2.2,"Iris-virginica"  
7.7,2.6,6.9,2.3,"Iris-virginica"  
6.0,2.2,5.0,1.5,"Iris-virginica"  
6.9,3.2,5.7,2.3,"Iris-virginica"  
5.6,2.8,4.9,2.0,"Iris-virginica"  
7.7,2.8,6.7,2.0,"Iris-virginica"  
6.3,2.7,4.9,1.8,"Iris-virginica"  
6.7,3.3,5.7,2.1,"Iris-virginica"  
7.2,3.2,6.0,1.8,"Iris-virginica"  
6.2,2.8,4.8,1.8,"Iris-virginica"  
6.1,3.0,4.9,1.8,"Iris-virginica"  
6.4,2.8,5.6,2.1,"Iris-virginica"  
7.2,3.0,5.8,1.6,"Iris-virginica"  
7.4,2.8,6.1,1.9,"Iris-virginica"  
7.9,3.8,6.4,2.0,"Iris-virginica"  
6.4,2.8,5.6,2.2,"Iris-virginica"  
6.3,2.8,5.1,1.5,"Iris-virginica"  
6.1,2.6,5.6,1.4,"Iris-virginica"  
7.7,3.0,6.1,2.3,"Iris-virginica"  
6.3,3.4,5.6,2.4,"Iris-virginica"  
6.4,3.1,5.5,1.8,"Iris-virginica"  
6.0,3.0,4.8,1.8,"Iris-virginica"  
6.9,3.1,5.4,2.1,"Iris-virginica"  
6.7,3.1,5.6,2.4,"Iris-virginica"  
6.9,3.1,5.1,2.3,"Iris-virginica"  
5.8,2.7,5.1,1.9,"Iris-virginica"  
6.8,3.2,5.9,2.3,"Iris-virginica"  
6.7,3.3,5.7,2.5,"Iris-virginica"  
6.7,3.0,5.2,2.3,"Iris-virginica"  
6.3,2.5,5.0,1.9,"Iris-virginica"  
6.5,3.0,5.2,2.0,"Iris-virginica"  
6.3,3.4,5.4,2.3,"Iris-virginica"
```

```
6.2,3.4,3.4,2.3, Iris-virginica"
5.9,3.0,5.1,1.8,"Iris-virginica"
5.1 3.5 1.4 0.2 Iris-setosa
5.4 3.9 1.7 0.4 Iris-setosa
5.4 3.7 1.5 0.2 Iris-setosa
5.8 4.0 1.2 0.2 Iris-setosa
5.7 4.4 1.5 0.4 Iris-setosa
5.4 3.9 1.3 0.4 Iris-setosa
5.1 3.5 1.4 0.3 Iris-setosa
5.7 3.8 1.7 0.3 Iris-setosa
5.1 3.8 1.5 0.3 Iris-setosa
5.4 3.4 1.7 0.2 Iris-setosa
5.1 3.7 1.5 0.4 Iris-setosa
5.1 3.3 1.7 0.5 Iris-setosa
5.2 3.5 1.5 0.2 Iris-setosa
5.2 3.4 1.4 0.2 Iris-setosa
5.4 3.4 1.5 0.4 Iris-setosa
5.2 4.1 1.5 0.1 Iris-setosa
5.5 4.2 1.4 0.2 Iris-setosa
5.5 3.5 1.3 0.2 Iris-setosa
5.1 3.4 1.5 0.2 Iris-setosa
5.1 3.8 1.9 0.4 Iris-setosa
5.1 3.8 1.6 0.2 Iris-setosa
5.3 3.7 1.5 0.2 Iris-setosa
7.0 3.2 4.7 1.4 Iris-versicolor
6.4 3.2 4.5 1.5 Iris-versicolor
6.9 3.1 4.9 1.5 Iris-versicolor
5.5 2.3 4.0 1.3 Iris-versicolor
6.5 2.8 4.6 1.5 Iris-versicolor
5.7 2.8 4.5 1.3 Iris-versicolor
6.3 3.3 4.7 1.6 Iris-versicolor
6.6 2.9 4.6 1.3 Iris-versicolor
5.2 2.7 3.9 1.4 Iris-versicolor
5.9 3.0 4.2 1.5 Iris-versicolor
6.0 2.2 4.0 1.0 Iris-versicolor
6.1 2.9 4.7 1.4 Iris-versicolor
5.6 2.9 3.6 1.3 Iris-versicolor
6.7 3.1 4.4 1.4 Iris-versicolor
5.6 3.0 4.5 1.5 Iris-versicolor
5.8 2.7 4.1 1.0 Iris-versicolor
6.2 2.2 4.5 1.5 Iris-versicolor
```

5.6 2.5 3.9 1.1 Iris-versicolor
5.9 3.2 4.8 1.8 Iris-versicolor
6.1 2.8 4.0 1.3 Iris-versicolor
6.3 2.5 4.9 1.5 Iris-versicolor
6.1 2.8 4.7 1.2 Iris-versicolor
6.4 2.9 4.3 1.3 Iris-versicolor
6.6 3.0 4.4 1.4 Iris-versicolor
6.8 2.8 4.8 1.4 Iris-versicolor
6.7 3.0 5.0 1.7 Iris-versicolor
6.0 2.9 4.5 1.5 Iris-versicolor
5.7 2.6 3.5 1.0 Iris-versicolor
5.5 2.4 3.8 1.1 Iris-versicolor
5.5 2.4 3.7 1.0 Iris-versicolor
5.8 2.7 3.9 1.2 Iris-versicolor
6.0 2.7 5.1 1.6 Iris-versicolor
5.4 3.0 4.5 1.5 Iris-versicolor
6.0 3.4 4.5 1.6 Iris-versicolor
6.7 3.1 4.7 1.5 Iris-versicolor
6.3 2.3 4.4 1.3 Iris-versicolor
5.6 3.0 4.1 1.3 Iris-versicolor
5.5 2.5 4.0 1.3 Iris-versicolor
5.5 2.6 4.4 1.2 Iris-versicolor
6.1 3.0 4.6 1.4 Iris-versicolor
5.8 2.6 4.0 1.2 Iris-versicolor
5.6 2.7 4.2 1.3 Iris-versicolor
5.7 3.0 4.2 1.2 Iris-versicolor
5.7 2.9 4.2 1.3 Iris-versicolor
6.2 2.9 4.3 1.3 Iris-versicolor
5.1 2.5 3.0 1.1 Iris-versicolor
5.7 2.8 4.1 1.3 Iris-versicolor
6.3 3.3 6.0 2.5 Iris-virginica
5.8 2.7 5.1 1.9 Iris-virginica
7.1 3.0 5.9 2.1 Iris-virginica
6.3 2.9 5.6 1.8 Iris-virginica
6.5 3.0 5.8 2.2 Iris-virginica
7.6 3.0 6.6 2.1 Iris-virginica
7.3 2.9 6.3 1.8 Iris-virginica
6.7 2.5 5.8 1.8 Iris-virginica
7.2 3.6 6.1 2.5 Iris-virginica
6.5 3.2 5.1 2.0 Iris-virginica
6.4 2.7 5.3 1.9 Iris-virginica

```
6.8 3.0 5.5 2.1 Iris-virginica
5.7 2.5 5.0 2.0 Iris-virginica
5.8 2.8 5.1 2.4 Iris-virginica
6.4 3.2 5.3 2.3 Iris-virginica
6.5 3.0 5.5 1.8 Iris-virginica
7.7 3.8 6.7 2.2 Iris-virginica
7.7 2.6 6.9 2.3 Iris-virginica
6.0 2.2 5.0 1.5 Iris-virginica
6.9 3.2 5.7 2.3 Iris-virginica
5.6 2.8 4.9 2.0 Iris-virginica
7.7 2.8 6.7 2.0 Iris-virginica
6.3 2.7 4.9 1.8 Iris-virginica
6.7 3.3 5.7 2.1 Iris-virginica
7.2 3.2 6.0 1.8 Iris-virginica
6.2 2.8 4.8 1.8 Iris-virginica
6.1 3.0 4.9 1.8 Iris-virginica
6.4 2.8 5.6 2.1 Iris-virginica
7.2 3.0 5.8 1.6 Iris-virginica
7.4 2.8 6.1 1.9 Iris-virginica
7.9 3.8 6.4 2.0 Iris-virginica
6.4 2.8 5.6 2.2 Iris-virginica
6.3 2.8 5.1 1.5 Iris-virginica
6.1 2.6 5.6 1.4 Iris-virginica
7.7 3.0 6.1 2.3 Iris-virginica
6.3 3.4 5.6 2.4 Iris-virginica
6.4 3.1 5.5 1.8 Iris-virginica
6.0 3.0 4.8 1.8 Iris-virginica
6.9 3.1 5.4 2.1 Iris-virginica
6.7 3.1 5.6 2.4 Iris-virginica
6.9 3.1 5.1 2.3 Iris-virginica
5.8 2.7 5.1 1.9 Iris-virginica
6.8 3.2 5.9 2.3 Iris-virginica
6.7 3.3 5.7 2.5 Iris-virginica
6.7 3.0 5.2 2.3 Iris-virginica
6.3 2.5 5.0 1.9 Iris-virginica
6.5 3.0 5.2 2.0 Iris-virginica
6.2 3.4 5.4 2.3 Iris-virginica
5.9 3.0 5.1 1.8 Iris-virginica
Sql compiled:
CREATE TABLE tmp_pyodps_xxxx LIFECYCLE 1 AS
```

```
SELECT *
FROM xxx.`pyodps_iris` t1
WHERE t1.`sepallength` > 5
Instance ID: 2019xxxx
Log view: http://logview.odps.aliyun.com/logview/?h=http://service.cn.maxcompute.aliyun.com/
api&p=xxxx&i=2019xxxx&token=xxxx
```

	sepallength	sepalwidth	petallength	petalwidth	name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	5.4	3.9	1.7	0.4	Iris-setosa
2	5.4	3.7	1.5	0.2	Iris-setosa
3	5.8	4.0	1.2	0.2	Iris-setosa
4	5.7	4.4	1.5	0.4	Iris-setosa

Sql compiled:

```
CREATE TABLE tmp_pyodps_xxxx LIFECYCLE 1 AS
```

```
SELECT *
FROM xxxx.`pyodps_iris` t1
WHERE (t1.`sepallength` < 5) AND (t1.`petallength` > 1.5)
```

Instance ID: 2019xxxx

Log view: <http://logview.odps.aliyun.com/logview/?h=http://service.cn.maxcompute.aliyun.com/api&p=xxxx&i=2019xxxx&token=xxxx>

	sepallength	sepalwidth	petallength	petalwidth	name
0	4.8	3.4	1.6	0.2	Iris-setosa
1	4.8	3.4	1.9	0.2	Iris-setosa
2	4.7	3.2	1.6	0.2	Iris-setosa
3	4.8	3.1	1.6	0.2	Iris-setosa
4	4.9	2.4	3.3	1.0	Iris-versicolor

6.9.3. Use a PyODPS node to pass parameters

This topic describes how to use a PyODPS node to pass parameters.

Prerequisites

The following operations are completed:

- [Activate MaxCompute](#)
- [Activate DataWorks](#)
- Create a workflow in the DataWorks console. In this example, create a workflow in a DataWorks workspace in basic mode. For more information, see [Create a workflow](#).

Procedure

1. Prepare test data.

- i. Create a partitioned table and a source table, and import data to the source table. For more information, see [Create tables and import data](#). In this example, use the following table creation statements and source data.

- Execute the following statement to create a partitioned table named *user_detail*:

```
create table if not exists user_detail
(
  userid BIGINT comment 'user ID',
  job STRING comment 'job type',
  education STRING comment 'education level'
) comment 'user information table'
partitioned by (dt STRING comment 'date', region STRING comment 'region');
```

- Execute the following statement to create a source table named *user_detail_ods*:

```
create table if not exists user_detail_ods
(
  userid BIGINT comment 'user ID',
  job STRING comment 'job type',
  education STRING comment 'education level',
  dt STRING comment 'date',
  region STRING comment 'region'
);
```

- Create a source data file named *user_detail.txt* and save the following data to the file. Import the data to the *user_detail_ods* table.

```
0001,internet,bachelor,20190715,beijing
0002,education,junior college,20190716,beijing
0003,finance,master,20190715,shandong
0004,internet,master,20190715,beijing
```

- ii. Log on to the DataWorks console and click Workspaces in the left-side navigation pane. On the Workspaces page, find the target workspace and click Data Analytics in the Actions column. On the Data Development tab, right-click the target workflow in the Business process section and choose **New > MaxCompute > ODPS SQL**.
- iii. In the New node dialog box, set the **Node name** parameter and click **Submit**.
- iv. On the configuration tab of the ODPS SQL node, enter the following code in the code editor:

```
insert overwrite table user_detail partition (dt,region)
select userid,job,education,dt,region from user_detail_ods;
```

- v. Click the Run icon in the toolbar to insert the data from the *user_detail_ods* table to the *user_detail* table.

□

2. Use a PyODPS node to pass parameters.

- i. Log on to the [DataWorks console](#).
- ii. In the left-side navigation pane, click **Workspaces**.
- iii. On the Workspaces page, find the target workspace and click **Data Analytics** in the Actions column.
- iv. On the Data Development tab, right-click the target workflow in the Business process section and choose **New > MaxCompute > PyODPS 2**.
- v. In the New node dialog box, set the **Node name** parameter and click **Submit**.
- vi. On the configuration tab of the PyODPS 2 node, enter the following code in the code editor:

```
import sys
reload(sys)
print('dt=' + args['dt'])
# Set UTF-8 as the default encoding format.
sys.setdefaultencoding('utf8')
# Obtain the user_detail table.
t = o.get_table('user_detail')
# Receive the partition field that is passed.
with t.open_reader(partition='dt=' + args['dt'] + ',region=beijing') as reader1:
    count = reader1.count
print("Query data in the partitioned table:")
for record in reader1:
    print record[0],record[1],record[2]
```

- vii. Click **Advanced run (run with parameters)** in the toolbar.
- viii. In the Parameters dialog box, set the parameters and click **Confirm**. Parameter description:
 - **Scheduling Resource Group**: Set this parameter to **Common scheduler resource group**.
 - **dt**: Set this parameter to **dt=20190715**.
- ix. View the running result of the PyODPS 2 node on the **Run Log** tab.



6.9.4. Use a PyODPS node to read data from a partitioned table

This topic describes how to use a PyODPS node to read data from a partitioned table.

read data from a partitioned table

Prerequisites

The following operations are completed:

- [Activate MaxCompute](#)

- **Activate DataWorks**
- Create a workflow in the DataWorks console. In this example, create a workflow in a DataWorks workspace in basic mode. For more information, see [Create a workflow](#).

Procedure

1. Prepare test data.

- Create a partitioned table and a source table, and import data to the source table. For more information, see [Create tables and import data](#). In this example, use the following table creation statements and source data.

- Execute the following statement to create a partitioned table named *user_detail*:

```
create table if not exists user_detail
(
  userid  BIGINT comment 'user ID',
  job     STRING comment 'job type',
  education STRING comment 'education level'
) comment 'user information table'
partitioned by (dt STRING comment 'date', region STRING comment 'region');
```

- Execute the following statement to create a source table named *user_detail_ods*:

```
create table if not exists user_detail_ods
(
  userid  BIGINT comment 'user ID',
  job     STRING comment 'job type',
  education STRING comment 'education level',
  dt      STRING comment 'date',
  region  STRING comment 'region'
);
```

- Create a source data file named *user_detail.txt* and save the following data to the file. Import the data to the *user_detail_ods* table.

```
0001,Internet,bachelor,20190715,beijing
0002,education,junior college,20190716,beijing
0003,finance,master,20190715,shandong
0004,Internet,master,20190715,beijing
```

- Log on to the DataWorks console and click Workspaces in the left-side navigation pane. On the Workspaces page, find the target workspace and click Data Analytics in the Actions column. On the Data Development tab, right-click the target workflow in the Business process section and choose **New > MaxCompute > ODPS SQL**.
- In the New node dialog box, set the **Node name** parameter and click **Submit**.

- iv. On the configuration tab of the ODPS SQL node, enter the following code in the code editor:

```
insert overwrite table user_detail partition (dt,region)
select userid,job,education,dt,region from user_detail_ods;
```

- v. Click the Run icon in the toolbar to insert the data from the `user_detail_ods` table to the `user_detail` table.

□

2. Use a PyODPS node to read data from a partitioned table.

- i. Log on to the [DataWorks console](#).
- ii. In the left-side navigation pane, click **Workspaces**.
- iii. On the Workspaces page, find the target workspace and click **Data Analytics** in the Actions column.
- iv. On the Data Development tab, right-click the target workflow in the Business process section and choose **New > MaxCompute > PyODPS 2**.
- v. In the New node dialog box, set the **Node name** parameter and click **Submit**.
- vi. On the configuration tab of the PyODPS 2 node, enter the following code in the code editor:

```

import sys
from odps import ODPS
reload(sys)
print('dt=' + args['dt'])
# Set UTF-8 as the default encoding format.
sys.setdefaultencoding('utf8')
# Obtain the partitioned table.
t = o.get_table('user_detail')
# Check whether the specified partition exists.
print t.exist_partition('dt=20190715,region=beijing')
# View all partitions in the partitioned table.
for partition in t.partitions:
    print partition.name
# You can use one of the following methods to query data in the partitioned table:
# Method 1
with t.open_reader(partition='dt=20190715,region=beijing') as reader1:
    count = reader1.count
print("Query data in the partitioned table by using Method 1:")
for record in reader1:
    print record[0],record[1],record[2]
# Method 2
print("Query data in the partitioned table by using Method 2:")
reader2 = t.open_reader(partition='dt=20190715,region=beijing')
for record in reader2:
    print record["userid"],record["job"],record["education"]
# Method 3
print("Query data in the partitioned table by using Method 3:")
for record in o.read_table('user_detail', partition='dt=20190715,region=beijing'):
    print record["userid"],record["job"],record["education"]

```

vii. Click **Advanced run (run with parameters)** in the toolbar.

□

viii. In the **Parameters** dialog box, set the parameters and click **Confirm**. Parameter description:

- **Scheduling Resource Group:** Set this parameter to **Common scheduler resource group**.
- **dt:** Set this parameter to **dt=20190715**.

ix. View the running result of the PyODPS 2 node on the **Run Log** tab.

6.9.5. Use a PyODPS node to read data from the level-1 partition of the specified table

This topic describes how to use a PyODPS node to read data from the level-1 partition of the specified table.

Prerequisites

The following operations are completed:

- [Activate MaxCompute](#)
- [Activate DataWorks](#)
- Create a workflow in the DataWorks console. In this example, create a workflow in a DataWorks workspace in basic mode. For more information, see [Create a workflow](#).

Procedure

1. Prepare test data.

- Create a partitioned table and a source table, and import data to the source table. For more information, see [Create tables and import data](#). In this example, use the following table creation statements and source data.

- Execute the following statement to create a partitioned table named *user_detail*:

```
create table if not exists user_detail
(
  userid  BIGINT comment 'user ID',
  job     STRING comment 'job type',
  education STRING comment 'education level'
) comment 'user information table'
partitioned by (dt STRING comment 'date', region STRING comment 'region');
```

- Execute the following statement to create a source table named *user_detail_ods*:

```
create table if not exists user_detail_ods
(
  userid  BIGINT comment 'user ID',
  job     STRING comment 'job type',
  education STRING comment 'education level',
  dt      STRING comment 'date',
  region  STRING comment 'region'
);
```

- Create a source data file named *user_detail.txt* and save the following data to the file. Import the data to the *user_detail_ods* table.

```
0001,internet,bachelor,20190715,beijing
0002,education,junior college,20190716,beijing
0003,finance,master,20190715,shandong
0004,internet,master,20190715,beijing
```

- ii. Log on to the DataWorks console and click Workspaces in the left-side navigation pane. On the Workspaces page, find the target workspace and click Data Analytics in the Actions column. On the Data Development tab, right-click the target workflow in the Business process section and choose **New > MaxCompute > ODPS SQL**.
 - iii. In the New node dialog box, set the **Node name** parameter and click **Submit**.
 - iv. On the configuration tab of the ODPS SQL node, enter the following code in the code editor:

```
insert overwrite table user_detail partition (dt,region)
select userid,job,education,dt,region from user_detail_ods;
```
 - v. Click the Run icon in the toolbar to insert the data from the *user_detail_ods* table to the *user_detail* table.
-
2. Use a PyODPS node to read data from the level-1 partition of the *user_detail* table.
- i. Log on to the **DataWorks console**.
 - ii. In the left-side navigation pane, click **Workspaces**.
 - iii. On the Workspaces page, find the target workspace and click **Data Analytics** in the Actions column.
 - iv. On the Data Development tab, right-click the target workflow in the Business process section and choose **New > MaxCompute > PyODPS 2**.
 - v. In the New node dialog box, set the **Node name** parameter and click **Submit**.

- vi. On the configuration tab of the PyODPS 2 node, enter the following code in the code editor:

```
import sys
reload(sys)
# Set UTF-8 as the default encoding format.
sys.setdefaultencoding('utf8')

# Read data from the level-1 partition in asynchronous mode.
instance = o.run_sql('select * from user_detail WHERE dt=\'20190715\'')
instance.wait_for_success()
for record in instance.open_reader():
    print record["userid"],record["job"],record["education"]

# Read data from the level-1 partition in synchronous mode.
with o.execute_sql('select * from user_detail WHERE dt=\'20190715\'').open_reader() as reader
4:
    print reader4.raw
    for record in reader4:
        print record["userid"],record["job"],record["education"]

# Use the PyODPS DataFrame to read data from the level-1 partition.
pt_df = DataFrame(o.get_table('user_detail').get_partition('dt=20190715'))
print pt_df.head(10)
```

- vii. Click the Run icon in the toolbar.

□

- viii. View the running result of the PyODPS 2 node on the Run Log tab.

□

6.9.6. Use a PyODPS node to perform sequence operations

This topic describes how to use a PyODPS node to perform sequence operations.

Prerequisites

The following operations are completed:

- [Activate MaxCompute](#)
- [Activate DataWorks](#)
- Create a workflow in the DataWorks console. In this example, create a workflow in a DataWorks workspace in basic mode. For more information, see [Create a workflow](#).

Procedure

1. Create a table and import data to it.

- i. Download the **Iris** dataset named *iris.data* and rename it *iris.csv*.
- ii. Create a table named `pyodps_iris` and upload the *iris.csv* dataset. For more information, see [Create tables and import data](#). In this example, execute the following statement to create the `pyodps_iris` table:

```
CREATE TABLE if not exists pyodps_iris
(
  sepallength DOUBLE comment 'sepal length (cm)',
  sepalwidth  DOUBLE comment 'sepal width (cm)',
  petallength DOUBLE comment 'petal length (cm)',
  petalwidth  DOUBLE comment 'petal width (cm)',
  name        STRING comment 'type'
);
```

2. Log on to the **DataWorks console**.
3. In the left-side navigation pane, click **Workspaces**.
4. On the page that appears, find the workspace that you want to manage and click **Data Analytics** in the Actions column.
5. On the DataStudio page, right-click a workflow and choose **New > MaxCompute > PyODPS 2**.
6. In the **New node** dialog box, specify the **Node name** parameter and click **Submit**.
7. On the configuration tab of the PyODPS 2 node, enter the code of the node in the code editor. In this example, enter the following code:

```

from odps import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))

# Obtain data from the specified column.
print iris.sepalength.head(5)

print iris['sepalength'].head(5)

# View the data type of the column.
print iris.sepalength.dtype

# Change the data type of the column.
iris.sepalength.astype('int')

# Group values in the specified column and obtain the maximum value in each group.
print iris.groupby('name').sepalength.max().head(5)

print iris.sepalength.max()

# Rename the specified column.
print iris.sepalwidth.rename('speal_width').head(5)

# Obtain the sum of the values of the specified columns, place the sum in a new column, and rename the new column.
print (iris.sepalength + iris.sepalwidth).rename('sum_sepal').head(5)

```

8. Click the Run icon in the toolbar.

9. View the running result of the PyODPS 2 node on the Run Log tab. In this example, the following information appears on the Run Log tab:

Executing user script with PyODPS 0.8.0

Try to fetch data from tunnel

sepalength

0 4.9

1 4.7

2 4.6

3 5.0

4 5.4

Try to fetch data from tunnel

sepalength

0 4.9

```

0      4.7
1      4.6
2      5.0
3      5.4
FLOAT64
Sql compiled:
CREATE TABLE tmp_pyodps_ed78e3ba_f13c_4a49_812d_2790d57c25dd LIFECYCLE 1 AS
SELECT MAX(t1.`sepallength`) AS `sepallength_max`
FROM data_service_fr.`pyodps_iris` t1
GROUP BY t1.`name`

    sepallength_max
0      5.8
1      7.0
2      7.9
Collection: ref_0
odps.Table
  name: data_service_fr.`pyodps_iris`
  schema:
    sepallength      : double    # Sepal length (cm)
    sepalwidth       : double    # Sepal width (cm)
    petallength      : double    # Petal length (cm)
    petalwidth       : double    # Petal width (cm)
    name             : string    # Type
max = Max[float64]
sepallength = Column[sequence(float64)] 'sepallength' from collection ref_0
Try to fetch data from tunnel
    speal_width
0      3.0
1      3.2
2      3.1
3      3.6
4      3.9
Sql compiled:
CREATE TABLE tmp_pyodps_28120275_8d0f_4683_8318_302fa21459ac LIFECYCLE 1 AS
SELECT t1.`sepallength` + t1.`sepalwidth` AS `sum_sepal`
FROM data_service_fr.`pyodps_iris` t1

    sum_sepal
0      7.9

```



```
1    7.9
2    7.7
3    8.6
4    9.3
2019-08-13 10:48:13 INFO =====
=====
2019-08-13 10:48:13 INFO Exit code of the Shell command 0
2019-08-13 10:48:13 INFO --- Invocation of Shell command completed ---
2019-08-13 10:48:13 INFO Shell run successfully!
```

10. Use the same method to create and run another PyODPS node named PyExecute. On the configuration tab of the PyExecute node, enter the following code in the code editor:

```
from odps import options
from odps import DataFrame

# View the logview of the running instance.
options.verbose = True
iris = DataFrame(o.get_table('pyodps_iris'))
iris[iris.sepalwidth < 5].exclude('sepalwidth')[5].execute()

my_logs = []
def my_loggers(x):
    my_logs.append(x)

options.verbose_log = my_loggers

iris[iris.sepalwidth < 5].exclude('sepalwidth')[5].execute()

print(my_logs)

# Cache the intermediate collection.
cached = iris[iris.sepalwidth < 3.5].cache()
print cached.head(3)

# Concurrently execute methods in asynchronous mode.
from odps.df import Delay
delay = Delay() # Create a Delay object.
df = iris[iris.sepalwidth < 5].cache() # The sum(), mean(), and max() methods depend on cache().
future1 = df.sepalwidth.sum().execute(delay=delay) # This method immediately returns a future object without execution after it is called.
future2 = df.sepalwidth.mean().execute(delay=delay)
future3 = df.sepalwidth.max().execute(delay=delay)

delay.execute(n_parallel=3)

print future1.result()
print future2.result()
print future3.result()
```

In this example, the following logs are recorded after the PyExecute node is run:

Executing user script with PyODPS 0.8.0

Sql compiled:

```
CREATE TABLE tmp_pyodps_4a204590_0510_4e9c_823b_5b837a437840 LIFECYCLE 1 AS
```

```
SELECT t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name`
```

```
FROM data_service_fr.`pyodps_iris` t1
```

```
WHERE t1.`sepallength` < 5
```

```
LIMIT 5
```

Instance ID: 20190813025233386g04djssa

Log view: <http://logview.odps.aliyun.com/logview/XXX>

```
['Sql compiled:', 'CREATE TABLE tmp_pyodps_03b92c55_8442_4e61_8978_656495487b8a LIFECYCLE 1
AS \nSELECT t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name` \nFROM data_service_fr.`
pyodps_iris` t1 \nWHERE t1.`sepallength` < 5 \nLIMIT 5', 'Instance ID: 20190813025236282gcsna5pr2'
, u'
```

Log view: <http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.com/api&XXX>

	sepallength	sepalwidth	petallength	petalwidth	name
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa

454.6

3.05100671141

4.4

2019-08-13 10:52:48 INFO =====
=====

2019-08-13 10:52:48 INFO Exit code of the Shell command 0

2019-08-13 10:52:48 INFO --- Invocation of Shell command completed ---

2019-08-13 10:52:48 INFO Shell run successfully!

2019-08-13 10:52:48 INFO Current task status: FINISH

6.9.7. Use a PyODPS node to aggregate data

This topic describes how to use a PyODPS node to aggregate data.

Prerequisites

- **MaxCompute is activated.**
- **DataWorks is activated.**
- A workflow is created in the DataWorks console. In this example, a workflow is created for a DataWorks workspace in basic mode. For more information, see [Create a workflow](#).

Procedure

1. Create a table and import data to it.

- i. Download the **Iris** dataset named *iris.data* and rename it *iris.csv*.
- ii. Create a table named `pyodps_iris` and upload the *iris.csv* dataset. For more information, see [Create tables and import data](#). In this example, execute the following statement to create the `pyodps_iris` table:

```
CREATE TABLE if not exists pyodps_iris
(
  sepallength DOUBLE comment 'sepal length (cm)',
  sepalwidth  DOUBLE comment 'sepal width (cm)',
  petallength DOUBLE comment 'petal length (cm)',
  petalwidth  DOUBLE comment 'petal width (cm)',
  name        STRING comment 'type'
);
```

2. Log on to the [DataWorks console](#).
3. In the left-side navigation pane, click **Workspaces**.
4. On the DataStudio page, right-click a workflow and choose **New > MaxCompute > PyODPS 2**.
5. In the **New node** dialog box, specify the **Node name** parameter and click **Submit**.
6. On the configuration tab of the PyODPS 2 node, enter the code of the node in the code editor. In this example, enter the following code:

```
from odps.df import DataFrame
from odps.df import Delay
from odps.df import Scalar
from odps.df import agg
import numpy as np
import pandas as pd
iris = DataFrame(o.get_table('pyodps_iris'))

# Call the describe method to return the number, maximum value, minimum value, average value,
# and standard deviation of the numeric columns in the PyODPS DataFrame.
print iris.describe().execute()

# Obtain the maximum value from the specified column.
print iris.sepallength.max().execute()

# Remove duplicate columns from the PyODPS DataFrame.
print iris.name.unique().cat(sep=',').execute()

# Calculate the total number of rows in the PyODPS DataFrame.
print iris.count().execute()
```

```
# Aggregate data by group.
print iris.groupby('name').agg(iris.sepalength.max(),smin=iris.sepalength.min()).head()
print iris.groupby('name').agg(count=iris.name.count()).sort('count', ascending=False).head(5)
print iris.groupby('name').petallength.sum().head()
print iris.groupby('name').agg(iris.petallength.notnull().sum()).head()
print iris.groupby(Scalar(1)).petallength.sum().head()
```

```
# Aggregate data in a single column.
```

```
class Agg(object):
```

```
    def buffer(self):
        return [0.0, 0]
```

```
    def __call__(self, buffer, val):
        buffer[0] += val
        buffer[1] += 1
```

```
    def merge(self, buffer, pBuffer):
        buffer[0] += pBuffer[0]
        buffer[1] += pBuffer[1]
```

```
    def getvalue(self, buffer):
        if buffer[1] == 0:
            return 0.0
        return buffer[0] / buffer[1]
```

```
print iris.sepalwidth.agg(Agg).execute()
```

```
# Aggregate data in two columns.
```

```
class Agg2(object):
```

```
    def buffer(self):
        return [0.0, 0.0]
```

```
    def __call__(self, buffer, val1, val2):
        buffer[0] += val1
        buffer[1] += val2
```

```
    def merge(self, buffer, pBuffer):
```

```

    buffer[0] += pBuffer[0]
    buffer[1] += pBuffer[1]

def getvalue(self, buffer):
    if buffer[1] == 0:
        return 0.0
    return buffer[0] - buffer[1]

to_agg = agg([iris.sepalwidth, iris.sepalwidth], Agg2, rtype='float') # Call a user-defined aggregate function to aggregate data in two columns.
print iris.groupby('name').agg(val=to_agg).execute()

# Use HyperLogLog to approximate the number of distinct values.

df5 = DataFrame(pd.DataFrame({'a': np.random.randint(100000, size=100000)}))
print df5.a.hll_count().execute()

```

7. Click the Run icon in the toolbar.

□

8. View the running result of the PyODPS 2 node on the Run Log tab. In this example, the following information appears on the Run Log tab:

Executing user script with PyODPS 0.8.0

Sql compiled:

```

CREATE TABLE tmp_pyodps_28b83337_87f9_4b11_a084_f1896758d53b LIFECYCLE 1 AS
SELECT pyodps_udf_1567394092_9467b7ba_3121_4105_81dc_431e8e68be39(t3.`sepalwidth_count`,
t3.`sepalwidth_mean`, t3.`sepalwidth_std`, t3.`sepalwidth_min`, t3.`sepalwidth_quantile_25`, t
3.`sepalwidth_quantile_50`, t3.`sepalwidth_quantile_75`, t3.`sepalwidth_max`, t3.`sepalwidth_c
ount`, t3.`sepalwidth_mean`, t3.`sepalwidth_std`, t3.`sepalwidth_min`, t3.`sepalwidth_quantile_25
`, t3.`sepalwidth_quantile_50`, t3.`sepalwidth_quantile_75`, t3.`sepalwidth_max`, t3.`petalwidth_c
ount`, t3.`petalwidth_mean`, t3.`petalwidth_std`, t3.`petalwidth_min`, t3.`petalwidth_quantile_
25`, t3.`petalwidth_quantile_50`, t3.`petalwidth_quantile_75`, t3.`petalwidth_max`, t3.`petalwid
th_count`, t3.`petalwidth_mean`, t3.`petalwidth_std`, t3.`petalwidth_min`, t3.`petalwidth_quantile_
25`, t3.`petalwidth_quantile_50`, t3.`petalwidth_quantile_75`, t3.`petalwidth_max`) AS (`type`, `se
palwidth`, `petalwidth`, `petalwidth`)
FROM (
    SELECT COUNT(t2.`sepalwidth`) AS `sepalwidth_count`, AVG(t2.`sepalwidth`) AS `sepalwidth_
mean`, STDDEV_SAMP(t2.`sepalwidth`) AS `sepalwidth_std`, MIN(t2.`sepalwidth`) AS `sepalwidth_
min`, PERCENTILE(t2.`sepalwidth`, 0.25) AS `sepalwidth_quantile_25`, PERCENTILE(t2.`sepalwid
th`, 0.5) AS `sepalwidth_quantile_50`, PERCENTILE(t2.`sepalwidth`, 0.75) AS `sepalwidth_quanti
le_75`, MAX(t2.`sepalwidth`) AS `sepalwidth_max`, COUNT(t2.`sepalwidth`) AS `sepalwidth_coun
t`, AVG(t2.`sepalwidth`) AS `sepalwidth_mean`, STDDEV_SAMP(t2.`sepalwidth`) AS `sepalwidth_st

```

```

t, AVG(t2.`sepalwidth`) AS `sepalwidth_mean`, STDDEV_SAMP(t2.`sepalwidth`) AS `sepalwidth_std`, MIN(t2.`sepalwidth`) AS `sepalwidth_min`, PERCENTILE(t2.`sepalwidth`, 0.25) AS `sepalwidth_quantile_25`, PERCENTILE(t2.`sepalwidth`, 0.5) AS `sepalwidth_quantile_50`, PERCENTILE(t2.`sepalwidth`, 0.75) AS `sepalwidth_quantile_75`, MAX(t2.`sepalwidth`) AS `sepalwidth_max`, COUNT(t2.`petallength`) AS `petallength_count`, AVG(t2.`petallength`) AS `petallength_mean`, STDDEV_SAMP(t2.`petallength`) AS `petallength_std`, MIN(t2.`petallength`) AS `petallength_min`, PERCENTILE(t2.`petallength`, 0.25) AS `petallength_quantile_25`, PERCENTILE(t2.`petallength`, 0.5) AS `petallength_quantile_50`, PERCENTILE(t2.`petallength`, 0.75) AS `petallength_quantile_75`, MAX(t2.`petallength`) AS `petallength_max`, COUNT(t2.`petalwidth`) AS `petalwidth_count`, AVG(t2.`petalwidth`) AS `petalwidth_mean`, STDDEV_SAMP(t2.`petalwidth`) AS `petalwidth_std`, MIN(t2.`petalwidth`) AS `petalwidth_min`, PERCENTILE(t2.`petalwidth`, 0.25) AS `petalwidth_quantile_25`, PERCENTILE(t2.`petalwidth`, 0.5) AS `petalwidth_quantile_50`, PERCENTILE(t2.`petalwidth`, 0.75) AS `petalwidth_quantile_75`, MAX(t2.`petalwidth`) AS `petalwidth_max`
FROM (
  SELECT t1.`sepallength`, t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`
  FROM DQC_0221_dev.`pyodps_iris` t1
) t2
) t3

```

	type	sepallength	sepalwidth	petallength	petalwidth
0	count	149.000000	149.000000	149.000000	149.000000
1	mean	5.848322	3.051007	3.774497	1.205369
2	std	0.828594	0.433499	1.759651	0.761292
3	min	4.300000	2.000000	1.000000	0.100000
4	quantile_25	5.000000	2.000000	1.000000	0.000000
5	quantile_50	5.000000	3.000000	4.000000	1.000000
6	quantile_75	6.000000	3.000000	5.000000	1.000000
7	max	7.900000	4.400000	6.900000	2.500000

Sql compiled:

```

SELECT MAX(t2.`sepallength`) AS `sepallength_max`
FROM (
  SELECT t1.`sepallength`
  FROM DQC_0221_dev.`pyodps_iris` t1
) t2

```

7.9

Sql compiled:

```

SELECT WM_CONCAT(DISTINCT ',', t2.`name`) AS `name_cat`
FROM (

```

```
SELECT t1.`name`
FROM DQC_0221_dev.`pyodps_iris` t1
) t2
```

```
Iris-setosa,Iris-versicolor,Iris-virginica
149
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_4b00d671_82b1_4cd8_8af4_f11dbdac4570 LIFECYCLE 1 AS
SELECT t1.`name`, MAX(t1.`sepallength`) AS `sepallength_max`, MIN(t1.`sepallength`) AS `smin`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY t1.`name`
```

	name	sepallength_max	smin
0	Iris-setosa	5.8	4.3
1	Iris-versicolor	7.0	4.9
2	Iris-virginica	7.9	4.9

Sql compiled:

```
CREATE TABLE tmp_pyodps_276c901f_7cf0_48ee_aaae_0d3092822b34 LIFECYCLE 1 AS
SELECT t1.`name`, COUNT(t1.`name`) AS `count`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY t1.`name`
ORDER BY count DESC
LIMIT 10000
```

	name	count
0	Iris-versicolor	50
1	Iris-virginica	50
2	Iris-setosa	49

Sql compiled:

```
CREATE TABLE tmp_pyodps_440d12ec_8496_4b87_b33f_b9624941e5bd LIFECYCLE 1 AS
SELECT SUM(t1.`petallength`) AS `petallength_sum`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY t1.`name`
```

Instance ID: 20190902031604255go6c472m

	petallength_sum
0	71.8
1	213.0


```
2      277.6
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_61e62f5f_7b4f_4971_8c39_dcacba37764 LIFECYCLE 1 AS
SELECT t1.`name`, SUM(IF(t1.`petallength` IS NOT NULL, 1, 0)) AS `petallength_sum`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY t1.`name`
Instance ID: 20190902031607441ghzyr392
```

	name	petallength_sum
0	Iris-setosa	49
1	Iris-versicolor	50
2	Iris-virginica	50

Sql compiled:

```
CREATE TABLE tmp_pyodps_daeffd50_88fe_4b4d_9db9_57810cfe71c3 LIFECYCLE 1 AS
SELECT SUM(t1.`petallength`) AS `petallength_sum`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY 1
```

	petallength_sum
0	562.4

Sql compiled:

```
SELECT pyodps_udf_1567394173_ef78183a_a4f5_4372_91dd_18d6ecc6f651(t2.`sepalwidth`) AS `sepalwidth_aggregation`
FROM (
  SELECT t1.`sepalwidth`
  FROM DQC_0221_dev.`pyodps_iris` t1
) t2
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_e312f110_64e7_465e_adaf_fd053f125a90 LIFECYCLE 1 AS
SELECT t1.`name`, pyodps_udf_1567394220_49231d7a_faeb_4f2f_9de5_36c4aefaa29b(t1.`sepalwidth`, t1.`sepalwidth`) AS `val`
FROM DQC_0221_dev.`pyodps_iris` t1
GROUP BY t1.`name`
```

	name	val
0	Iris-setosa	77.0

```
0  Iris-setosa -11.8
1  Iris-versicolor -158.3
2  Iris-virginica -180.7

63021
```

6.9.8. Use a PyODPS node to perform column operations

This topic describes how to use a PyODPS node to perform column operations.

Prerequisites

- **MaxCompute is activated.**
- **DataWorks is activated.**
- A workflow is created in the DataWorks console. In this example, a workflow is created for a DataWorks workspace in basic mode. For more information, see [Create a workflow](#).

Procedure

1. Create a table and import data to it.
 - i. Download the **Iris** dataset named *iris.data* and rename it *iris.csv*.
 - ii. Create a table named `pyodps_iris` and upload the *iris.csv* dataset. For more information, see [Create tables and import data](#). In this example, execute the following statement to create the `pyodps_iris` table:

```
CREATE TABLE if not exists pyodps_iris
(
  sepalength DOUBLE comment 'sepal length (cm)',
  sepalwidth DOUBLE comment 'sepal width (cm)',
  petallength DOUBLE comment 'petal length (cm)',
  petalwidth DOUBLE comment 'petal width (cm)',
  name STRING comment 'type'
);
```

2. Log on to the [DataWorks console](#).
3. In the left-side navigation pane, click **Workspaces**.
4. On the page that appears, find the workspace that you want to manage and click **Data Analytics** in the Actions column.
5. On the DataStudio page, right-click a workflow and choose **New > MaxCompute > PyODPS 2**.
6. In the **New node** dialog box, specify the **Node name** parameter and click **Submit**.
7. On the configuration tab of the PyODPS 2 node, enter the code of the node in the code editor. In this example, enter the following code:

```
from odps import DataFrame
```

```

import numpy as np
import pandas as pd

iris = DataFrame(o.get_table('pyodps_iris'))

# Check for NULL values.
print iris.sepalength.isnull().head(5)

# Perform logical judgment.
print (iris.sepalength > 5).ifelse('gt5','lte5').rename('cmp5').head(5)

# Perform judgment based on multiple conditions.
print iris.sepalength.switch(4.9,'eq4.9',5.0,'eq5.0',default='noeq').rename('equalness').head(5)
from odps.df import switch
print switch(iris.sepalength == 4.9,'eq4.9',iris.sepalength == 5.0,'eq5.0',default='noeq').rename('equalness').head(5)

# Change some values in the specified column.
iris[iris.sepalength > 5,'cmp5'] = 'gt5'
iris[iris.sepalength <=5,'cmp5'] = 'lte5'
print iris.head(5)

# Perform mathematical calculation.
print (iris.sepalength * 10).log().head(5)
fields = [iris.sepalength,(iris.sepalength /2).rename('sepalength/2'),(iris.sepalength ** 2).rename('Square of sepalength')]
print iris[fields].head(5)
print (iris.sepalength < 5).head(5)

# Perform operations on collections.
data = {'id': [1,2], 'a': [['a1','b1'],['c1']], 'b': [{'a2': 0, 'b2': 1, 'c2': 2},{'d2': 3, 'e2': 4}]}
df = pd.DataFrame(data)
print df

df1 = DataFrame(df, unknown_as_string=True, as_type={'a': 'list<string>','b': 'dict<string,int64>'})
print df1.dtypes
print df1.head()

```

```

print df1[df1.id,df1.a[0],df1.b.len()].head()
print df1.a.explode().head()
print df1.a.explode(pos=True).head()
print df1.b.explode().head()
print df1.b.explode(['key','value']).head()

# Return the execution result of the explode method together with the specified columns.
print df1[df1.id,df1.a.explode()].head()

#isin,notin,cut.

# Call the isin method to check whether elements in the specified sequence are in a specific collection. The notin method can check whether elements in the specified sequence is not in a specific collection.
print iris.sepalength.isin([4.9,5.1]).rename('sepalength').head()

# Call the cut method to specify ranges and return the range to which each element in the specified sequence belongs.
print iris.sepalength.cut(range(6),labels=['0-1','1-2','2-3','3-4','4-5']).rename('sepalength_cut').head(5)

# You can use the include_under parameter to specify whether to return values that are less than the specified upper limit. You can use the include_over parameter to specify whether to return values that are greater than the specified lower limit.
labels = ['0-1', '1-2', '2-3', '3-4', '4-5', '5-']
iris.sepalength.cut(range(6), labels=labels, include_over=True).rename('sepalength_cut').head(5)
)

```

8. Click the Run icon in the toolbar.

□

9. View the running result of the PyODPS 2 node on the Run Log tab. In this example, the following information appears on the Run Log tab:

Executing user script with PyODPS 0.8.0

Sql compiled:

```

CREATE TABLE tmp_pyodps_32e39a9a_8ae6_4815_b465_4b9dfa3bf1b9 LIFECYCLE 1 AS
SELECT t1.`sepalength` IS NULL AS `sepalength`
FROM DQC_0221_dev.`pyodps_iris` t1

```

sepalength

0 False

```
1 False
```

```
2 False
```

```
3 False
```

```
4 False
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_d684aded_8ad8_4119_8558_1fddd774dd3f LIFECYCLE 1 AS
```

```
SELECT IF(t1.`sepallength` > 5, 'gt5', 'lte5') AS `cmp5`
```

```
FROM DQC_0221_dev.`pyodps_iris` t1
```

```
    cmp5
```

```
0 lte5
```

```
1 lte5
```

```
2 lte5
```

```
3 lte5
```

```
4 gt5
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_cfc5bdb9_6d3e_4711_8acc_913b29b96c9a LIFECYCLE 1 AS
```

```
SELECT CASE t1.`sepallength` WHEN 4.9 THEN 'eq4.9' WHEN 5.0 THEN 'eq5.0' ELSE 'noeq' END AS `equalness`
```

```
FROM DQC_0221_dev.`pyodps_iris` t1
```

```
    equalness
```

```
0 eq4.9
```

```
1 noeq
```

```
2 noeq
```

```
3 eq5.0
```

```
4 noeq
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_d1fd70b4_2ad8_4dc9_838e_f566f7f5841e LIFECYCLE 1 AS
```

```
SELECT CASE WHEN t1.`sepallength` == 4.9 THEN 'eq4.9' WHEN t1.`sepallength` == 5.0 THEN 'eq5.0' ELSE 'noeq' END AS `equalness`
```

```
FROM DQC_0221_dev.`pyodps_iris` t1
```

```
    equalness
```

```
0 eq4.9
```

```
1 noeq
```

```
2 noeq
```

```
3 eq5.0
```

```
4 noeq
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553 LIFECYCLE 1 AS
```

```
SELECT t1.`sepallength`, t1.`sepalwidth`, t1.`petallength`, t1.`petalwidth`, t1.`name`, IF(t1.`sepallength` <= 5, 'lte5', IF(t1.`sepallength` > 5, 'gt5', CAST(NULL AS STRING))) AS `cmp5`
FROM DQC_0221_dev.`pyodps_iris` t1
```

	sepallength	sepalwidth	petallength	petalwidth	name	cmp5
0	4.9	3.0	1.4	0.2	Iris-setosa	lte5
1	4.7	3.2	1.3	0.2	Iris-setosa	lte5
2	4.6	3.1	1.5	0.2	Iris-setosa	lte5
3	5.0	3.6	1.4	0.2	Iris-setosa	lte5
4	5.4	3.9	1.7	0.4	Iris-setosa	gt5

Sql compiled:

```
CREATE TABLE tmp_pyodps_4bfb0aa0_7ec5_4f35_9480_d36baf2b8079 LIFECYCLE 1 AS
SELECT LN(t1.`sepallength` * 10) AS `sepallength`
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
```

	sepallength
0	3.891820
1	3.850148
2	3.828641
3	3.912023
4	3.988984

Sql compiled:

```
CREATE TABLE tmp_pyodps_deb6fa45_fbb1_41d2_8174_76d35eebea9c LIFECYCLE 1 AS
SELECT t1.`sepallength`, t1.`sepallength` / 2 AS `sepallength/2`, POW(t1.`sepallength`, 2) AS `Square of sepallength`
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
```

	sepallength	sepallength/2	Square of sepallength
0	4.9	2.45	24.01
1	4.7	2.35	22.09
2	4.6	2.30	21.16
3	5.0	2.50	25.00
4	5.4	2.70	29.16

Sql compiled:

```
CREATE TABLE tmp_pyodps_fa326546_c5e8_466f_915d_e783b7dcacdf LIFECYCLE 1 AS
SELECT t1.`sepallength` < 5 AS `sepallength`
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
```

	sepallength
0	True

```

0 True
1 True
2 True
3 False
4 False
    a          b id
0 [a1, b1] {u'c2': 2, u'a2': 0, u'b2': 1} 1
1 [c1]      {u'd2': 3, u'e2': 4} 2
odps.Schema {
  a list<string>
  b dict<string,int64>
  id int64
}
    a          b id
0 [a1, b1] {u'c2': 2, u'a2': 0, u'b2': 1} 1
1 [c1]      {u'd2': 3, u'e2': 4} 2
  id a b
0 1 a1 3
1 2 c1 2
    a
0 a1
1 b1
2 c1
  a_pos a
0 0 a1
1 1 b1
2 0 c1
  b_key b_value
0 c2 2
1 a2 0
2 b2 1
3 d2 3
4 e2 4
  key value
0 c2 2
1 a2 0
2 b2 1
3 d2 3
4 e2 4

  id a

```

```
0 1 a1
```

```
1 1 b1
```

```
2 2 c1
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_fb28d9b9_fa45_4712_a564_dd751641286e LIFECYCLE 1 AS
```

```
SELECT t1.`sepallength` IN (4.9, 5.1) AS `sepallength`
```

```
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
```

sepallength

```
0 True
```

```
1 False
```

```
2 False
```

```
3 False
```

```
4 False
```

```
5 False
```

```
6 False
```

```
7 False
```

```
8 True
```

```
9 False
```

```
10 False
```

```
11 False
```

```
12 False
```

```
13 False
```

```
14 False
```

```
15 False
```

```
16 True
```

```
17 False
```

```
18 True
```

```
19 False
```

```
20 True
```

```
21 False
```

```
22 True
```

```
23 False
```

```
24 False
```

```
25 False
```

```
26 False
```

```
27 False
```

```
28 False
```

```
29 False
```

```
30 False
```

```
31 False
```



```

32  False
33  True
34  False
35  False
36  True
37  False
38  True
39  False
40  False
41  False
42  False
43  True
44  False
45  True
46  False
47  False
48  False
49  False
50  False
51  False
52  False
53  False
54  False
55  False
56  True
57  False
58  False
59  False

```

Sql compiled:

```

CREATE TABLE tmp_pyodps_1f85e8e1_a9f0_473f_ac0e_d526fdd80dc4 LIFECYCLE 1 AS
SELECT CASE WHEN (0 < t1.`sepallength`) AND (t1.`sepallength` <= 1) THEN '0-1' WHEN (1 < t1.`sep
allength`) AND (t1.`sepallength` <= 2) THEN '1-2' WHEN (2 < t1.`sepallength`) AND (t1.`sepallength
` <= 3) THEN '2-3' WHEN (3 < t1.`sepallength`) AND (t1.`sepallength` <= 4) THEN '3-4' WHEN (4 < t1.`
sepallength`) AND (t1.`sepallength` <= 5) THEN '4-5' END AS `sepallength_cut`
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1
  sepallength_cut
0      4-5
1      4-5
2      4-5
3      4-5
4      None

```

```

4      NONE
Sql compiled:
CREATE TABLE tmp_pyodps_080a677e_904b_4edc_a961_160f8c664d7f LIFECYCLE 1 AS
SELECT CASE WHEN (0 < t1.`sepallength`) AND (t1.`sepallength` <= 1) THEN '0-1' WHEN (1 < t1.`sep
allength`) AND (t1.`sepallength` <= 2) THEN '1-2' WHEN (2 < t1.`sepallength`) AND (t1.`sepallength
` <= 3) THEN '2-3' WHEN (3 < t1.`sepallength`) AND (t1.`sepallength` <= 4) THEN '3-4' WHEN (4 < t1.`
sepallength`) AND (t1.`sepallength` <= 5) THEN '4-5' WHEN 5 < t1.`sepallength` THEN '5-' END AS `s
epallength_cut`
FROM DQC_0221_dev.`tmp_pyodps_d9e5d64e_1f21_4ffb_993f_70d3f36b6553` t1

```

6.9.9. Use a PyODPS node to sort data

This topic describes how to use a PyODPS node to sort data.

Prerequisites

- **MaxCompute is activated.**
- **DataWorks is activated.**
- A workflow is created in the DataWorks console. In this example, a workflow is created for a DataWorks workspace in basic mode. For more information, see [Create a workflow](#).

Procedure

1. Create a table and import data to it.
 - i. Download the **Iris** dataset named *iris.data* and rename it *iris.csv*.
 - ii. Create a table named `pyodps_iris` and upload the *iris.csv* dataset. For more information, see [Create tables and import data](#). In this example, execute the following statement to create the `pyodps_iris` table:

```

CREATE TABLE if not exists pyodps_iris
(
  sepallength DOUBLE comment 'sepal length (cm)',
  sepalwidth  DOUBLE comment 'sepal width (cm)',
  petallength DOUBLE comment 'petal length (cm)',
  petalwidth  DOUBLE comment 'petal width (cm)',
  name        STRING comment 'type'
);

```

2. Log on to the [DataWorks console](#).
3. In the left-side navigation pane, click **Workspaces**.
4. On the page that appears, find the workspace that you want to manage and click **Data Analytics** in the Actions column.
5. On the DataStudio page, right-click a workflow and choose **New > MaxCompute > PyODPS 2**.
6. In the **New node** dialog box, specify the **Node name** parameter and click **Submit**.
7. On the configuration tab of the PyODPS 2 node, enter the following code in the code editor:

```

from odps.df import DataFrame

iris = DataFrame(o.get_table('pyodps_iris'))

# Sort data.

print iris.sort('sepalwidth').head(5)

# You can use one of the following methods to sort data in descending order:

# Set the ascending parameter to False.
print iris.sort('sepalwidth',ascending=False).head(5)

# Use a hyphen (-) as the sorting parameter.
print iris.sort(-iris.sepalwidth).head(5)

# Sort multiple fields.
print iris.sort(['sepalwidth','petallength']).head(5)

# To sort multiple fields in different orders, you can set the ascending parameter to a list of BOOLEAN values. The number of values in the list must be the same as the number of fields to be sorted.
print iris.sort(['sepalwidth','petallength'],ascending=[True,False]).head(5)
print iris.sort(['sepalwidth',-iris.petallength]).head(5)

```

8. Click the Run icon in the toolbar.

9. View the running result of the PyODPS 2 node on the Run Log tab.



In this example, the following information appears on the Run Log tab:

```

Sql compiled:
CREATE TABLE tmp_pyodps_d1b06785_dc18_4288_ad34_de860de1be08 LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth
LIMIT 10000
Instance ID: 20191010061554817gwml0lim

```

	sepalwidth	sepalwidth	petallength	petalwidth	name
0	5.0	2.0	3.5	1.0	Iris-versicolor
1	6.0	2.2	5.0	1.5	Iris-virginica

```

2      6.2      2.2      4.5      1.5 Iris-versicolor
3      6.0      2.2      4.0      1.0 Iris-versicolor
4      5.5      2.3      4.0      1.3 Iris-versicolor

```

Sql compiled:

```

CREATE TABLE tmp_pyodps_3cb90bb2_fb95_43fb_ae84_f2b5a27d72dc LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth DESC
LIMIT 10000
Instance ID: 20191010061601287gs086792

```

	sepalwidth	sepalwidth	petalwidth	petalwidth	name
0	5.7	4.4	1.5	0.4	Iris-setosa
1	5.5	4.2	1.4	0.2	Iris-setosa
2	5.2	4.1	1.5	0.1	Iris-setosa
3	5.8	4.0	1.2	0.2	Iris-setosa
4	5.4	3.9	1.3	0.4	Iris-setosa

Sql compiled:

```

CREATE TABLE tmp_pyodps_97b080bb_e014_48e8_a310_4b45fcd6a2ed LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth DESC
LIMIT 10000
Instance ID: 20191010061606927g6emz192

```

	sepalwidth	sepalwidth	petalwidth	petalwidth	name
0	5.7	4.4	1.5	0.4	Iris-setosa
1	5.5	4.2	1.4	0.2	Iris-setosa
2	5.2	4.1	1.5	0.1	Iris-setosa
3	5.8	4.0	1.2	0.2	Iris-setosa
4	5.4	3.9	1.3	0.4	Iris-setosa

Sql compiled:

```

CREATE TABLE tmp_pyodps_6fe37b6e_6705_4052_b733_211eb9bd16ac LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth, petalwidth
LIMIT 10000
Instance ID: 20191010061611714gn586792

```

	sepalwidth	sepalwidth	petalwidth	petalwidth	name

```

0    5.0    2.0    3.5    1.0 Iris-versicolor
1    6.0    2.2    4.0    1.0 Iris-versicolor
2    6.2    2.2    4.5    1.5 Iris-versicolor
3    6.0    2.2    5.0    1.5 Iris-virginica
4    4.5    2.3    1.3    0.3  Iris-setosa

```

Sql compiled:

```

CREATE TABLE tmp_pyodps_a52c805c_94a1_4a75_a6af_4fc9ed06ae68 LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth, petallength DESC
LIMIT 10000

```

Instance ID: 20191010061616553gw3m9592

	sepalwidth	petallength	petalwidth	name
0	5.0	2.0	3.5	1.0 Iris-versicolor
1	6.0	2.2	5.0	1.5 Iris-virginica
2	6.2	2.2	4.5	1.5 Iris-versicolor
3	6.0	2.2	4.0	1.0 Iris-versicolor
4	6.3	2.3	4.4	1.3 Iris-versicolor

Sql compiled:

```

CREATE TABLE tmp_pyodps_aac5538e_9b40_4078_b3c6_852b99c663c1 LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pyodps_iris` t1
ORDER BY sepalwidth, petallength DESC
LIMIT 10000

```

Instance ID: 20191010061621329gvmkc292

	sepalwidth	petallength	petalwidth	name
0	5.0	2.0	3.5	1.0 Iris-versicolor
1	6.0	2.2	5.0	1.5 Iris-virginica
2	6.2	2.2	4.5	1.5 Iris-versicolor
3	6.0	2.2	4.0	1.0 Iris-versicolor
4	6.3	2.3	4.4	1.3 Iris-versicolor

6.9.10. Use a PyODPS node to de-duplicate data

This topic describes how to use a PyODPS node to de-duplicate data.

Prerequisites

- **MaxCompute is activated.**

- **DataWorks is activated.**
- A workflow is created in the DataWorks console. In this example, a workflow is created for a DataWorks workspace in basic mode. For more information, see [Create a workflow](#).

Procedure

1. Create a table and import data to it.
 - i. Download the **Iris** dataset named *iris.data* and rename it *iris.csv*.
 - ii. Create a table named `pyodps_iris` and upload the *iris.csv* dataset. For more information, see [Create tables and import data](#). In this example, execute the following statement to create the `pyodps_iris` table:

```
CREATE TABLE if not exists pyodps_iris
(
  sepallength DOUBLE comment 'sepal length (cm)',
  sepalwidth  DOUBLE comment 'sepal width (cm)',
  petallength DOUBLE comment 'petal length (cm)',
  petalwidth  DOUBLE comment 'petal width (cm)',
  name        STRING comment 'type'
);
```

2. Log on to the [DataWorks console](#).
3. In the left-side navigation pane, click **Workspaces**.
4. On the page that appears, find the workspace that you want to manage and click **Data Analytics** in the Actions column.
5. On the DataStudio page, right-click a workflow and choose **New > MaxCompute > PyODPS 2**.
6. In the **New node** dialog box, specify the **Node name** parameter and click **Submit**.
7. On the configuration tab of the PyODPS 2 node, enter the code of the node in the code editor. In this example, enter the following code:

```
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
print iris[['name']].distinct()
print iris.distinct('name')
print iris.distinct('name','sepallength').head(3)
# You can call the unique method to de-duplicate data in the specified sequence. The sequence whose data is de-duplicated by using the unique method cannot be selected as a column.
print iris.name.unique()
```

8. Click the **Run** icon in the toolbar.
9. View the running result of the PyODPS 2 node on the **Run Log** tab.

In this example, the following information appears on the Run Log tab:

```
Sql compiled:
```

```
CREATE TABLE tmp_pyodps_ed85ebd5_d678_44dd_9ece_bff1822376f6 LIFECYCLE 1 AS
SELECT DISTINCT t1.`name`
FROM WB_BestPractice_dev.`pyodps_iris` t1
Instance ID: 2019101006391142g2cp5692
```

```
      name
0  Iris-setosa
1  Iris-versicolor
2  Iris-virginica
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_8ce6128f_9c6f_45af_b9de_c73ce9d5ba51 LIFECYCLE 1 AS
SELECT DISTINCT t1.`name`
FROM WB_BestPractice_dev.`pyodps_iris` t1
```

Instance ID: 20191010063915987gmuws592

```
      name
0  Iris-setosa
1  Iris-versicolor
2  Iris-virginica
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_a3dc338e_0fea_4d5f_847c_79fb19ec1c72 LIFECYCLE 1 AS
SELECT DISTINCT t1.`name`, t1.`sepalength`
FROM WB_BestPractice_dev.`pyodps_iris` t1
```

Instance ID: 2019101006392210gj056292

```
      name sepalength
0  Iris-setosa      4.3
1  Iris-setosa      4.4
2  Iris-setosa      4.5
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_bc0917bb_f10c_426b_9b75_47e94478382a LIFECYCLE 1 AS
SELECT t2.`name`
FROM (
  SELECT DISTINCT t1.`name`
  FROM WB_BestPractice_dev.`pyodps_iris` t1
```

```
) t2
```

```
Instance ID: 20191010063927189g9fsz192
```

```

      name
0  Iris-setosa
1  Iris-versicolor
2  Iris-virginica

```

6.9.11. Use a PyODPS node to sample data

This topic describes how to use a PyODPS node to sample data.

Prerequisites

- **MaxCompute is activated.**
- **DataWorks is activated.**
- A workflow is created in the DataWorks console. In this example, a workflow is created for a DataWorks workspace in basic mode. For more information, see [Create a workflow](#).

Procedure

1. Create a table and import data to it.
 - i. Download the **Iris** dataset named *iris.data* and rename it *iris.csv*.
 - ii. Create a table named `pyodps_iris` and upload the *iris.csv* dataset. For more information, see [Create tables and import data](#). In this example, execute the following statement to create the `pyodps_iris` table:

```

CREATE TABLE if not exists pyodps_iris
(
  sepallength DOUBLE comment 'sepal length (cm)',
  sepalwidth  DOUBLE comment 'sepal width (cm)',
  petallength DOUBLE comment 'petal length (cm)',
  petalwidth  DOUBLE comment 'petal width (cm)',
  name        STRING comment 'type'
);

```

2. Log on to the [DataWorks console](#).
3. In the left-side navigation pane, click **Workspaces**.
4. On the page that appears, find the workspace that you want to manage and click **Data Analytics** in the Actions column.
5. On the DataStudio page, right-click a workflow and choose **New > MaxCompute > PyODPS 2**.
6. In the **New node** dialog box, specify the **Node name** parameter and click **Submit**.
7. On the configuration tab of the PyODPS node, enter the code of the node in the code editor. In this example, enter the following code:


```

# Sample data.
from odps.df import DataFrame
iris = DataFrame(o.get_table('pyodps_iris'))
# Sample data by part.
print iris.sample(parts=10).head(5) # Divide the target data into 10 parts. Part 0 is obtained by default.
print iris.sample(parts=10,i=0).head(5) # Divide the target data into 10 parts and obtain part 0.
print iris.sample(parts=10,i=[2,5]).head(5) # Divide the target data into 10 parts and obtain part 2 and part 5.
print iris.sample(parts=10,columns=['name','sepalwidth']).head(5) # Divide the target data into 10 parts and sample data based on the values of name and sepalwidth.

# Sample data based on the specified number of data records or proportion.
print iris.sample(n=100).head() # Obtain 100 data records.

print iris.sample(frac=0.3).head() # Obtain 30% of the target data.

# Sample data based on the weight column.
print iris.sample(n=100,weights='sepallength').head()

print iris.sample(n=100,weights='sepalwidth',replace=True).head()

# Perform stratified sampling.
print iris.sample(strata='name',n={'Iris-setosa': 10,'Iris-versicolor': 10}).head()
print iris.sample(strata='name',frac={'Iris-setosa': 0.5,'Iris-versicolor': 0.4}).head()

```

8. Click the Run icon in the toolbar.



9. View the running result of the PyODPS 2 node on the Run Log tab. In this example, the following information appears on the Run Log tab:

```

Collection: ref_0
odps.Table
name: WB_BestPractice_dev.`pyodps_iris`
schema:
  sepallength      : double    # Sepal length (cm)
  sepalwidth       : double    # Sepal width (cm)
  petallength      : double    # Petal length (cm)
  petalwidth       : double    # Petal width (cm)
  name             : string    # Type
Sample[collection]

```

```

_input:
_parts:
  Scalar[int8]
  10
_i:
_replace:
  Scalar[boolean]
  False
Collection: ref_0
odps.Table
  name: WB_BestPractice_dev.`pyodps_iris`
  schema:
    sepalength      : double    # Sepal length (cm)
    sepalwidth      : double    # Sepal width (cm)
    petallength     : double    # Petal length (cm)
    petalwidth      : double    # Petal width (cm)
    name            : string    # Type
Sample[collection]
_input:
_parts:
  Scalar[int8]
  10
_i:
_replace:
  Scalar[boolean]
  False
Collection: ref_0
odps.Table
  name: WB_BestPractice_dev.`pyodps_iris`
  schema:
    sepalength      : double    # Sepal length (cm)
    sepalwidth      : double    # Sepal width (cm)
    petallength     : double    # Petal length (cm)
    petalwidth      : double    # Petal width (cm)
    name            : string    # Type
Sample[collection]
_input:
_parts:
  Scalar[int8]
  10
_i:

```

```

_replace:
  Scalar[boolean]
  False
Collection: ref_0
odps.Table
  name: WB_BestPractice_dev.`pyodps_iris`
  schema:
    sepallength      : double    # Sepal length (cm)
    sepalwidth       : double    # Sepal width (cm)
    petallength      : double    # Petal length (cm)
    petalwidth       : double    # Petal width (cm)
    name             : string    # Type
Sample[collection]
  _input:
  _parts:
    Scalar[int8]
    10
  _i:
  _sampled_fields:
    name = Column[sequence(string)] 'name' from collection ref_0
    sepalwidth = Column[sequence(float64)] 'sepalwidth' from collection ref_0
  _replace:
    Scalar[boolean]
    False
Executing RandomSample...
Command: PAI -name RandomSample -project algo_public -Dreplace="false" -Dlifecycle="1" -DoutputTableName="tmp_pyodps_1570690014_69f3d75d_9537_4c9c_87ea_a5f6ad8d2e07" -DsampleSize="100" -DinputTableName="WB_BestPractice_dev.pyodps_iris";
Instance ID: 20191010064654985g6co9592

Sub Instance: create_output (20191010064700688g5cbn62m_71ee0561_bcc4_4147_b849_f74688353fb6)

Sub Instance: without_replacement (20191010064703694g9cbn62m_93a8a15b_ffd1_4afe_8928_19f28455a15c)

Try to fetch data from tunnel

  sepallength sepalwidth petallength petalwidth      name
0      5.1      3.5      1.4      0.2  Iris-setosa

```

id	sepal	petal	width	height	name
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	4.6	3.4	1.4	0.3	Iris-setosa
6	4.4	2.9	1.4	0.2	Iris-setosa
7	4.9	3.1	1.5	0.1	Iris-setosa
8	4.8	3.4	1.6	0.2	Iris-setosa
9	4.8	3.0	1.4	0.1	Iris-setosa
10	4.3	3.0	1.1	0.1	Iris-setosa
11	5.1	3.5	1.4	0.3	Iris-setosa
12	5.7	3.8	1.7	0.3	Iris-setosa
13	5.1	3.8	1.5	0.3	Iris-setosa
14	5.1	3.7	1.5	0.4	Iris-setosa
15	4.6	3.6	1.0	0.2	Iris-setosa
16	5.1	3.3	1.7	0.5	Iris-setosa
17	4.8	3.4	1.9	0.2	Iris-setosa
18	5.0	3.4	1.6	0.4	Iris-setosa
19	5.2	3.5	1.5	0.2	Iris-setosa
20	5.2	3.4	1.4	0.2	Iris-setosa
21	4.8	3.1	1.6	0.2	Iris-setosa
22	5.2	4.1	1.5	0.1	Iris-setosa
23	5.5	4.2	1.4	0.2	Iris-setosa
24	4.9	3.1	1.5	0.1	Iris-setosa
25	5.0	3.2	1.2	0.2	Iris-setosa
26	4.4	3.0	1.3	0.2	Iris-setosa
27	5.1	3.4	1.5	0.2	Iris-setosa
28	5.0	3.5	1.3	0.3	Iris-setosa
29	4.5	2.3	1.3	0.3	Iris-setosa
..	...	...	...	...	...
70	7.1	3.0	5.9	2.1	Iris-virginica
71	7.6	3.0	6.6	2.1	Iris-virginica
72	7.3	2.9	6.3	1.8	Iris-virginica
73	7.2	3.6	6.1	2.5	Iris-virginica
74	6.5	3.2	5.1	2.0	Iris-virginica
75	6.8	3.0	5.5	2.1	Iris-virginica
76	5.8	2.8	5.1	2.4	Iris-virginica
77	7.7	3.8	6.7	2.2	Iris-virginica
78	7.7	2.6	6.9	2.3	Iris-virginica
79	7.7	2.8	6.7	2.0	Iris-virginica

80	6.3	2.7	4.9	1.8	Iris-virginica
81	6.7	3.3	5.7	2.1	Iris-virginica
82	6.2	2.8	4.8	1.8	Iris-virginica
83	6.1	3.0	4.9	1.8	Iris-virginica
84	6.4	2.8	5.6	2.1	Iris-virginica
85	7.2	3.0	5.8	1.6	Iris-virginica
86	7.4	2.8	6.1	1.9	Iris-virginica
87	7.9	3.8	6.4	2.0	Iris-virginica
88	6.3	2.8	5.1	1.5	Iris-virginica
89	6.3	3.4	5.6	2.4	Iris-virginica
90	6.4	3.1	5.5	1.8	Iris-virginica
91	6.0	3.0	4.8	1.8	Iris-virginica
92	6.9	3.1	5.4	2.1	Iris-virginica
93	6.9	3.1	5.1	2.3	Iris-virginica
94	5.8	2.7	5.1	1.9	Iris-virginica
95	6.8	3.2	5.9	2.3	Iris-virginica
96	6.7	3.3	5.7	2.5	Iris-virginica
97	6.3	2.5	5.0	1.9	Iris-virginica
98	6.2	3.4	5.4	2.3	Iris-virginica
99	5.9	3.0	5.1	1.8	Iris-virginica

[100 rows x 5 columns]

Executing RandomSample...

Command: PAI -name RandomSample -project algo_public -Dreplace="false" -DsampleRatio="0.3" -DoutputTableName="tmp_pyodps_1570690039_e1867332_72ea_4656_928d_3bd6e31d87c7" -Dlifecycle="1" -DinputTableName="WB_BestPractice_dev.pyodps_iris";

Instance ID: 20191010064720117gmpms38

Sub Instance: create_output (20191010064725740grcbn62m_b338a671_6047_4360_8792_41d2e748e41f)

Sub Instance: without_replacement (20191010064728747gtcbn62m_6c9914da_d5c3_4336_b076_163edb1bf48a)

Try to fetch data from tunnel

	sepalength	sepalwidth	petallength	petalwidth	name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	4.6	3.1	1.5	0.2	Iris-setosa
3	4.8	3.4	1.6	0.2	Iris-setosa
4	5.7	4.4	1.5	0.4	Iris-setosa
5	5.1	3.5	1.4	0.3	Iris-setosa

6	5.0	3.4	1.6	0.4	Iris-setosa
7	5.2	3.4	1.4	0.2	Iris-setosa
8	4.9	3.1	1.5	0.1	Iris-setosa
9	5.5	3.5	1.3	0.2	Iris-setosa
10	4.4	3.2	1.3	0.2	Iris-setosa
11	5.0	3.3	1.4	0.2	Iris-setosa
12	5.7	2.8	4.5	1.3	Iris-versicolor
13	5.2	2.7	3.9	1.4	Iris-versicolor
14	5.0	2.0	3.5	1.0	Iris-versicolor
15	5.6	2.9	3.6	1.3	Iris-versicolor
16	5.8	2.7	4.1	1.0	Iris-versicolor
17	6.1	2.8	4.0	1.3	Iris-versicolor
18	6.6	3.0	4.4	1.4	Iris-versicolor
19	6.8	2.8	4.8	1.4	Iris-versicolor
20	6.0	2.9	4.5	1.5	Iris-versicolor
21	5.4	3.0	4.5	1.5	Iris-versicolor
22	5.7	2.9	4.2	1.3	Iris-versicolor
23	5.7	2.8	4.1	1.3	Iris-versicolor
24	6.3	2.9	5.6	1.8	Iris-virginica
25	4.9	2.5	4.5	1.7	Iris-virginica
26	6.7	2.5	5.8	1.8	Iris-virginica
27	6.4	2.7	5.3	1.9	Iris-virginica
28	6.8	3.0	5.5	2.1	Iris-virginica
29	5.7	2.5	5.0	2.0	Iris-virginica
30	5.8	2.8	5.1	2.4	Iris-virginica
31	6.5	3.0	5.5	1.8	Iris-virginica
32	6.0	2.2	5.0	1.5	Iris-virginica
33	6.3	2.7	4.9	1.8	Iris-virginica
34	7.2	3.2	6.0	1.8	Iris-virginica
35	6.2	2.8	4.8	1.8	Iris-virginica
36	6.1	3.0	4.9	1.8	Iris-virginica
37	6.4	2.8	5.6	2.1	Iris-virginica
38	7.2	3.0	5.8	1.6	Iris-virginica
39	6.3	2.8	5.1	1.5	Iris-virginica
40	6.4	3.1	5.5	1.8	Iris-virginica
41	6.0	3.0	4.8	1.8	Iris-virginica
42	6.8	3.2	5.9	2.3	Iris-virginica
43	6.3	2.5	5.0	1.9	Iris-virginica
44	6.2	3.4	5.4	2.3	Iris-virginica

Executing WeightedSample

Executing WeightedSample...

Command: PAI -name WeightedSample -project algo_public -DinputTableName="WB_BestPractice_dev.pyodps_iris" -DsampleSize="100" -DprobCol="sepalength" -Dreplace="false" -DoutputTableName="tmp_pyodps_1570690063_6a62857e_8f85_4ea7_99ef_08aa259546d4" -Dlifecycle="1";

Instance ID: 20191010064743533gnpms38

Sub Instance: create_output (20191010064748787gkdbn62m_8d47bfb7_e470_4cce_8b69_28811f190083)

Sub Instance: without_replacement (20191010064751793gmdbn62m_230a1d26_5c2e_440e_a31d_fe9e63c6f906)

Try to fetch data from tunnel

	sepalength	sepalwidth	petallength	petalwidth	name
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa
2	5.0	3.6	1.4	0.2	Iris-setosa
3	5.4	3.9	1.7	0.4	Iris-setosa
4	5.0	3.4	1.5	0.2	Iris-setosa
5	4.4	2.9	1.4	0.2	Iris-setosa
6	4.8	3.4	1.6	0.2	Iris-setosa
7	4.8	3.0	1.4	0.1	Iris-setosa
8	5.4	3.9	1.3	0.4	Iris-setosa
9	5.1	3.5	1.4	0.3	Iris-setosa
10	5.7	3.8	1.7	0.3	Iris-setosa
11	4.6	3.6	1.0	0.2	Iris-setosa
12	5.0	3.4	1.6	0.4	Iris-setosa
13	5.2	3.5	1.5	0.2	Iris-setosa
14	5.2	3.4	1.4	0.2	Iris-setosa
15	4.7	3.2	1.6	0.2	Iris-setosa
16	4.8	3.1	1.6	0.2	Iris-setosa
17	5.5	4.2	1.4	0.2	Iris-setosa
18	4.9	3.1	1.5	0.1	Iris-setosa
19	5.0	3.2	1.2	0.2	Iris-setosa
20	5.5	3.5	1.3	0.2	Iris-setosa
21	4.9	3.1	1.5	0.1	Iris-setosa
22	5.1	3.4	1.5	0.2	Iris-setosa
23	4.5	2.3	1.3	0.3	Iris-setosa
24	4.8	3.0	1.4	0.3	Iris-setosa
25	5.1	3.8	1.6	0.2	Iris-setosa
26	4.6	3.2	1.4	0.2	Iris-setosa

```

27    5.3    3.7    1.5    0.2  Iris-setosa
28    5.0    3.3    1.4    0.2  Iris-setosa
29    7.0    3.2    4.7    1.4  Iris-versicolor
..    ...    ...    ...    ...    ...
70    7.2    3.6    6.1    2.5  Iris-virginica
71    6.4    2.7    5.3    1.9  Iris-virginica
72    5.7    2.5    5.0    2.0  Iris-virginica
73    5.8    2.8    5.1    2.4  Iris-virginica
74    6.4    3.2    5.3    2.3  Iris-virginica
75    7.7    3.8    6.7    2.2  Iris-virginica
76    6.9    3.2    5.7    2.3  Iris-virginica
77    5.6    2.8    4.9    2.0  Iris-virginica
78    7.7    2.8    6.7    2.0  Iris-virginica
79    6.3    2.7    4.9    1.8  Iris-virginica
80    6.7    3.3    5.7    2.1  Iris-virginica
81    7.2    3.2    6.0    1.8  Iris-virginica
82    6.2    2.8    4.8    1.8  Iris-virginica
83    6.1    3.0    4.9    1.8  Iris-virginica
84    7.2    3.0    5.8    1.6  Iris-virginica
85    7.9    3.8    6.4    2.0  Iris-virginica
86    6.4    2.8    5.6    2.2  Iris-virginica
87    6.3    2.8    5.1    1.5  Iris-virginica
88    6.1    2.6    5.6    1.4  Iris-virginica
89    6.3    3.4    5.6    2.4  Iris-virginica
90    6.4    3.1    5.5    1.8  Iris-virginica
91    6.0    3.0    4.8    1.8  Iris-virginica
92    6.9    3.1    5.1    2.3  Iris-virginica
93    5.8    2.7    5.1    1.9  Iris-virginica
94    6.8    3.2    5.9    2.3  Iris-virginica
95    6.7    3.3    5.7    2.5  Iris-virginica
96    6.7    3.0    5.2    2.3  Iris-virginica
97    6.5    3.0    5.2    2.0  Iris-virginica
98    6.2    3.4    5.4    2.3  Iris-virginica
99    5.9    3.0    5.1    1.8  Iris-virginica

```

[100 rows x 5 columns]

Executing WeightedSample...

Command: PAI -name WeightedSample -project algo_public -DinputTableName="WB_BestPractice_dev.pyodps_iris" -DsampleSize="100" -DprobCol="sepalwidth" -Dreplace="true" -DoutputTableName="tmp_pyodps_1570690082_f55e899c_3cb4_4eeb_ade4_b8cb79e018dc" -Dlifecycle="1";

Instance ID: 2019101006480392g9ers38

Sub Instance: create_output (20191010064808827g9ebn62m_fb70c859_913a_4830_9248_8c8eaf134f1d)

Sub Instance: with_replacement (20191010064811833gdebn62m_07544cc2_1d7d_4fa5_972d_196eb6b9f537)

	sepalength	sepalwidth	petallength	petalwidth	name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.6	3.4	1.4	0.3	Iris-setosa
2	5.0	3.4	1.5	0.2	Iris-setosa
3	5.0	3.4	1.5	0.2	Iris-setosa
4	4.4	2.9	1.4	0.2	Iris-setosa
5	4.8	3.4	1.6	0.2	Iris-setosa
6	5.8	4.0	1.2	0.2	Iris-setosa
7	5.8	4.0	1.2	0.2	Iris-setosa
8	5.1	3.5	1.4	0.3	Iris-setosa
9	5.1	3.5	1.4	0.3	Iris-setosa
10	5.1	3.5	1.4	0.3	Iris-setosa
11	5.1	3.7	1.5	0.4	Iris-setosa
12	4.6	3.6	1.0	0.2	Iris-setosa
13	4.8	3.4	1.9	0.2	Iris-setosa
14	5.0	3.0	1.6	0.2	Iris-setosa
15	5.0	3.4	1.6	0.4	Iris-setosa
16	5.2	3.4	1.4	0.2	Iris-setosa
17	4.8	3.1	1.6	0.2	Iris-setosa
18	4.8	3.1	1.6	0.2	Iris-setosa
19	5.4	3.4	1.5	0.4	Iris-setosa
20	5.4	3.4	1.5	0.4	Iris-setosa
21	5.4	3.4	1.5	0.4	Iris-setosa
22	5.2	4.1	1.5	0.1	Iris-setosa
23	5.2	4.1	1.5	0.1	Iris-setosa
24	5.5	4.2	1.4	0.2	Iris-setosa
25	5.0	3.2	1.2	0.2	Iris-setosa
26	4.9	3.1	1.5	0.1	Iris-setosa
27	4.4	3.0	1.3	0.2	Iris-setosa
28	5.1	3.4	1.5	0.2	Iris-setosa
29	5.0	3.5	1.3	0.3	Iris-setosa
..	...	...	...	...	...
70	5.6	2.7	4.2	1.3	Iris-versicolor
71	5.7	2.9	4.2	1.3	Iris-versicolor

71	5.7	2.9	4.2	1.5	Iris-versicolor
72	6.3	3.3	6.0	2.5	Iris-virginica
73	7.1	3.0	5.9	2.1	Iris-virginica
74	6.3	2.9	5.6	1.8	Iris-virginica
75	7.3	2.9	6.3	1.8	Iris-virginica
76	7.2	3.6	6.1	2.5	Iris-virginica
77	6.4	2.7	5.3	1.9	Iris-virginica
78	6.8	3.0	5.5	2.1	Iris-virginica
79	6.8	3.0	5.5	2.1	Iris-virginica
80	5.8	2.8	5.1	2.4	Iris-virginica
81	6.2	2.8	4.8	1.8	Iris-virginica
82	6.2	2.8	4.8	1.8	Iris-virginica
83	6.1	3.0	4.9	1.8	Iris-virginica
84	6.4	2.8	5.6	2.1	Iris-virginica
85	7.2	3.0	5.8	1.6	Iris-virginica
86	7.4	2.8	6.1	1.9	Iris-virginica
87	7.4	2.8	6.1	1.9	Iris-virginica
88	7.4	2.8	6.1	1.9	Iris-virginica
89	6.4	3.1	5.5	1.8	Iris-virginica
90	6.0	3.0	4.8	1.8	Iris-virginica
91	6.9	3.1	5.4	2.1	Iris-virginica
92	6.7	3.1	5.6	2.4	Iris-virginica
93	6.7	3.1	5.6	2.4	Iris-virginica
94	6.7	3.1	5.6	2.4	Iris-virginica
95	6.9	3.1	5.1	2.3	Iris-virginica
96	6.9	3.1	5.1	2.3	Iris-virginica
97	6.7	3.0	5.2	2.3	Iris-virginica
98	6.5	3.0	5.2	2.0	Iris-virginica
99	5.9	3.0	5.1	1.8	Iris-virginica

[100 rows x 5 columns]

Executing StratifiedSample...

Command: PAI -name StratifiedSample -project algo_public -Dlifecycle="1" -DoutputTableName="tmp_pyodps_1570690104_6cc52795_2a86_4634_a905_740b3a426d3f" -DsampleSize="Iris-setosa:10,Iris-versicolor:10" -DstrataColName="name" -DinputTableName="WB_BestPractice_dev.pyodps_iris";
Instance ID: 20191010064824633gaco9592

Sub Instance: create_output (20191010064829870g4fbn62m_dfa297c5_23b5_43f5_bb17_83ba8d263630)

Sub Instance: stratified_sampling (20191010064831874g7fbn62m_fc9eddb7_42f1_49fe_8206_891ef4)

51fb76)

Try to fetch data from tunnel

	sepalength	sepalwidth	petallength	petalwidth	name
0	5.4	3.9	1.7	0.4	Iris-setosa
1	4.3	3.0	1.1	0.1	Iris-setosa
2	5.4	3.9	1.3	0.4	Iris-setosa
3	5.1	3.3	1.7	0.5	Iris-setosa
4	4.7	3.2	1.6	0.2	Iris-setosa
5	4.5	2.3	1.3	0.3	Iris-setosa
6	5.0	3.5	1.6	0.6	Iris-setosa
7	5.1	3.8	1.9	0.4	Iris-setosa
8	4.8	3.0	1.4	0.3	Iris-setosa
9	5.0	3.3	1.4	0.2	Iris-setosa
10	7.0	3.2	4.7	1.4	Iris-versicolor
11	5.5	2.3	4.0	1.3	Iris-versicolor
12	6.5	2.8	4.6	1.5	Iris-versicolor
13	5.6	3.0	4.5	1.5	Iris-versicolor
14	5.7	2.6	3.5	1.0	Iris-versicolor
15	5.5	2.4	3.7	1.0	Iris-versicolor
16	5.0	2.3	3.3	1.0	Iris-versicolor
17	5.6	2.7	4.2	1.3	Iris-versicolor
18	5.7	3.0	4.2	1.2	Iris-versicolor
19	5.1	2.5	3.0	1.1	Iris-versicolor

Executing StratifiedSample...

Command: PAI -name StratifiedSample -project algo_public -DsampleRatio="Iris-setosa:0.5,Iris-versicolor:0.4" -DoutputTableName="tmp_pyodps_1570690128_a68477cd_19e5_4fe0_bb39_4712f76dd967" -Dlifecycle="1" -DstrataColName="name" -DinputTableName="WB_BestPractice_dev.pyodps_iris"

;

Instance ID: 20191010064848733gbers38

Sub Instance: create_output (20191010064853918gwfbn62m_4eb22c22_7051_4372_8d13_05c5a417aa87)

Sub Instance: stratified_sampling (20191010064855924g0gbn62m_b4242ac7_bd5a_47a8_a1f2_3367a6a101a7)

Try to fetch data from tunnel

	sepalength	sepalwidth	petallength	petalwidth	name
0	4.9	3.0	1.4	0.2	Iris-setosa
1	4.7	3.2	1.3	0.2	Iris-setosa

2	5.0	3.6	1.4	0.2	Iris-setosa
3	5.4	3.9	1.7	0.4	Iris-setosa
4	5.0	3.4	1.5	0.2	Iris-setosa
5	5.4	3.7	1.5	0.2	Iris-setosa
6	4.8	3.4	1.6	0.2	Iris-setosa
7	4.8	3.0	1.4	0.1	Iris-setosa
8	5.8	4.0	1.2	0.2	Iris-setosa
9	5.4	3.4	1.7	0.2	Iris-setosa
10	5.1	3.7	1.5	0.4	Iris-setosa
11	4.8	3.4	1.9	0.2	Iris-setosa
12	5.0	3.0	1.6	0.2	Iris-setosa
13	5.0	3.4	1.6	0.4	Iris-setosa
14	5.2	3.5	1.5	0.2	Iris-setosa
15	5.2	3.4	1.4	0.2	Iris-setosa
16	4.7	3.2	1.6	0.2	Iris-setosa
17	5.2	4.1	1.5	0.1	Iris-setosa
18	5.0	3.2	1.2	0.2	Iris-setosa
19	5.1	3.4	1.5	0.2	Iris-setosa
20	4.5	2.3	1.3	0.3	Iris-setosa
21	5.0	3.5	1.6	0.6	Iris-setosa
22	5.1	3.8	1.9	0.4	Iris-setosa
23	5.1	3.8	1.6	0.2	Iris-setosa
24	5.3	3.7	1.5	0.2	Iris-setosa
25	7.0	3.2	4.7	1.4	Iris-versicolor
26	6.4	3.2	4.5	1.5	Iris-versicolor
27	6.9	3.1	4.9	1.5	Iris-versicolor
28	6.5	2.8	4.6	1.5	Iris-versicolor
29	5.7	2.8	4.5	1.3	Iris-versicolor
30	6.6	2.9	4.6	1.3	Iris-versicolor
31	5.6	2.9	3.6	1.3	Iris-versicolor
32	5.6	3.0	4.5	1.5	Iris-versicolor
33	5.6	2.5	3.9	1.1	Iris-versicolor
34	6.1	2.8	4.7	1.2	Iris-versicolor
35	6.8	2.8	4.8	1.4	Iris-versicolor
36	5.5	2.4	3.8	1.1	Iris-versicolor
37	5.5	2.4	3.7	1.0	Iris-versicolor
38	6.0	2.7	5.1	1.6	Iris-versicolor
39	5.6	3.0	4.1	1.3	Iris-versicolor
40	5.5	2.6	4.4	1.2	Iris-versicolor
41	6.1	3.0	4.6	1.4	Iris-versicolor
42	5.7	2.6	4.2	1.3	Iris-versicolor

42	5.7	3.0	4.2	1.2 Iris-versicolor
43	5.7	2.9	4.2	1.3 Iris-versicolor
44	6.2	2.9	4.3	1.3 Iris-versicolor

6.9.12. Use a PyODPS node to scale data

This topic describes how to use a PyODPS node to scale data.

Prerequisites

- **MaxCompute is activated.**
- **DataWorks is activated.**
- A workflow is created in the DataWorks console. In this example, a workflow is created for a DataWorks workspace in basic mode. For more information, see [Create a workflow](#).

Procedure

1. Create a table and import data to it.
 - i. Download the **Iris** dataset named *iris.data* and rename it *iris.csv*.
 - ii. Create a table named `pyodps_iris` and upload the *iris.csv* dataset. For more information, see [Create tables and import data](#). In this example, execute the following statement to create the `pyodps_iris` table:

```
CREATE TABLE if not exists pyodps_iris
(
  sepallength DOUBLE comment 'sepal length (cm)',
  sepalwidth  DOUBLE comment 'sepal width (cm)',
  petallength DOUBLE comment 'petal length (cm)',
  petalwidth  DOUBLE comment 'petal width (cm)',
  name        STRING comment 'type'
);
```

2. Log on to the **DataWorks console**. Click Workspaces in the left-side navigation pane. On the Workspaces page, find the required workspace and click Data Analytics in the Actions column. The Data Development tab appears.
3. In the left-side navigation pane, click **Workspaces**.
4. On the DataStudio page, right-click a workflow and choose **New > MaxCompute > PyODPS 2**.
5. In the **New node** dialog box, specify the **Node name** parameter and click **Submit**.
6. On the configuration tab of the PyODPS 2 node, enter the code of the node in the code editor: In this example, enter the following code:

```
# Scale data.
df = DataFrame(o.get_table('pytable'))

# Call the min_max_scale method to normalize data.
print df.min_max_scale(columns=['fid']).head()

# Use the feature_range parameter to specify an output value range for the min_max_scale method. For example, to specify (-1,1) as the output value range, execute the following statement:
print df.min_max_scale(columns=['fid'],feature_range=(-1,1)).head()

# To keep the original values, set the preserve parameter to True. In this case, a new column that contains the scaled data is added. The new column is named in the format Original column name_scaled. You can change the suffix _scaled by using the suffix parameter.
print df.min_max_scale(columns=['fid'],preserve=True).head()

# Use the group parameter to specify a group column.
print df.min_max_scale(columns=['fid'],group=['name']).head()

# You can use the group parameter to specify one or more group columns for the std_scale method. Similar to the min_max_scale method, the std_scale method obtains the minimum and maximum values from the group columns to scale data.
print df.std_scale(columns=['fid']).head()
```

7. Click the Run icon in the toolbar.



8. View the running result of the PyODPS 2 node on the Run Log tab.

□

In this example, the following information appears on the Run Log tab:

```
Sql compiled:
CREATE TABLE tmp_pyodps_59cb5533_6dc3_4b64_9312_b2da29d70cee LIFECYCLE 1 AS
SELECT /*+mapjoin(t5)*/ t2.`name`, t2.`id`, (IF((t5.`fid_max_1570691133` - t5.`fid_min_1570691133`)
== 0, 0, (t2.`fid` - t5.`fid_min_1570691133`) / (t5.`fid_max_1570691133` - t5.`fid_min_1570691133`)) *
1) + 0 AS `fid`
FROM (
  SELECT t1.`name`, t1.`id`, t1.`fid`, 1 AS `idx_col_1570691133`
  FROM WB_BestPractice_dev.`pytable` t1
) t2
INNER JOIN
(
  SELECT 1 AS `idx_col_1570691133`, MIN(t4.`fid`) AS `fid_min_1570691133`, MAX(t4.`fid`) AS `fid_m
  AS `fid_max_1570691133`
```

```

ax_1570691139`

FROM (
  SELECT t3.`fid`
  FROM WB_BestPractice_dev.`pytable` t3
) t4
) t5
ON t2.`idx_col_1570691133` == t5.`idx_col_1570691133`
Instance ID: 2019101007053495gl8by72m

  name id    fid
0 name1  4  1.000000
1 name2  2  0.526316
2 name2  3  0.000000
3 name1  4  0.710526
4 name1  3  0.184211
5 name1  3  0.684211
Sql compiled:
CREATE TABLE tmp_pyodps_88a24967_3bdc_41ce_85d7_4bbd23e4ce01 LIFECYCLE 1 AS
SELECT /*+mapjoin(t5)*/ t2.`name`, t2.`id`, (IF((t5.`fid_max_1570691139` - t5.`fid_min_1570691139`)
== 0, 0, (t2.`fid` - t5.`fid_min_1570691139`) / (t5.`fid_max_1570691139` - t5.`fid_min_1570691139`)) *
2) + -1 AS `fid`
FROM (
  SELECT t1.`name`, t1.`id`, t1.`fid`, 1 AS `idx_col_1570691139`
  FROM WB_BestPractice_dev.`pytable` t1
) t2
INNER JOIN
(
  SELECT 1 AS `idx_col_1570691139`, MIN(t4.`fid`) AS `fid_min_1570691139`, MAX(t4.`fid`) AS `fid_m
ax_1570691139`
  FROM (
    SELECT t3.`fid`
    FROM WB_BestPractice_dev.`pytable` t3
  ) t4
) t5
ON t2.`idx_col_1570691139` == t5.`idx_col_1570691139`
Instance ID: 20191010070539772gjo56292

  name id    fid
0 name1  4  1.000000
1 name2  2  0.052632
2 name2  3 -1.000000

```

```

3 name1 4 0.421053
4 name1 3 -0.631579
5 name1 3 0.368421

Sql compiled:

CREATE TABLE tmp_pyodps_439117fc_9ef7_4086_899d_a5bf77d653e5 LIFECYCLE 1 AS
SELECT /*+mapjoin(t5)*/ t2.`name`, t2.`id`, t2.`fid`, (IF((t5.`fid_max_1570691146` - t5.`fid_min_1570691146`) == 0, 0, (t2.`fid` - t5.`fid_min_1570691146`) / (t5.`fid_max_1570691146` - t5.`fid_min_1570691146`)) * 1) + 0 AS `fid_scaled`
FROM (
  SELECT t1.`name`, t1.`id`, t1.`fid`, 1 AS `idx_col_1570691146`
  FROM WB_BestPractice_dev.`pytable` t1
) t2
INNER JOIN
(
  SELECT 1 AS `idx_col_1570691146`, MIN(t4.`fid`) AS `fid_min_1570691146`, MAX(t4.`fid`) AS `fid_max_1570691146`
  FROM (
    SELECT t3.`fid`
    FROM WB_BestPractice_dev.`pytable` t3
  ) t4
) t5
ON t2.`idx_col_1570691146` == t5.`idx_col_1570691146`

```

Instance ID: 20191010070546769g0c14f72

	name	id	fid	fid_scaled
0	name1	4	5.3	1.000000
1	name2	2	3.5	0.526316
2	name2	3	1.5	0.000000
3	name1	4	4.2	0.710526
4	name1	3	2.2	0.184211
5	name1	3	4.1	0.684211

```

Sql compiled:

CREATE TABLE tmp_pyodps_d3839b4b_1087_4d52_91f5_52763b72f272 LIFECYCLE 1 AS
SELECT /*+mapjoin(t3)*/ t1.`name`, t1.`id`, (IF((t3.`fid_max_1570691151` - t3.`fid_min_1570691151`) == 0, 0, (t1.`fid` - t3.`fid_min_1570691151`) / (t3.`fid_max_1570691151` - t3.`fid_min_1570691151`)) * 1) + 0 AS `fid`
FROM WB_BestPractice_dev.`pytable` t1
INNER JOIN
(
  SELECT t2.`name`, MAX(t2.`fid`) AS `fid_max_1570691151`, MIN(t2.`fid`) AS `fid_min_1570691151`

```



```
FROM WB_BestPractice_dev.`pytable` t2
GROUP BY t2.`name`
) t3
ON t1.`name` == t3.`name`
Instance ID: 20191010070551756gtf14f72
```

	name	id	fid
0	name1	4	1.000000
1	name2	2	1.000000
2	name2	3	0.000000
3	name1	4	0.645161
4	name1	3	0.000000
5	name1	3	0.612903

Sql compiled:

```
CREATE TABLE tmp_pyodps_1ea6e5b4_129f_4d1e_a6a7_410e08d77ae6 LIFECYCLE 1 AS
SELECT /*+mapjoin(t5)*/ t2.`name`, t2.`id`, IF(t5.`fid_std_1570691157` == 0, 0, (t2.`fid` - t5.`fid_mean_1570691157`) / t5.`fid_std_1570691157`) AS `fid`
FROM (
  SELECT t1.`name`, t1.`id`, t1.`fid`, 1 AS `idx_col_1570691157`
  FROM WB_BestPractice_dev.`pytable` t1
) t2
INNER JOIN
(
  SELECT 1 AS `idx_col_1570691157`, AVG(t4.`fid`) AS `fid_mean_1570691157`, STDDEV(t4.`fid`) AS `fid_std_1570691157`
  FROM (
    SELECT t3.`fid`
    FROM WB_BestPractice_dev.`pytable` t3
  ) t4
) t5
ON t2.`idx_col_1570691157` == t5.`idx_col_1570691157`
Instance ID: 20191010070557788gdl14f72
```

	name	id	fid
0	name1	4	1.436467
1	name2	2	0.026118
2	name2	3	-1.540938
3	name1	4	0.574587
4	name1	3	-0.992468
5	name1	3	0.406334

```
5 name1 3 0.496234
```

6.9.13. Use a PyODPS node to process NULL values

This topic describes how to use a PyODPS node to process NULL values.

Prerequisites

- [Activate MaxCompute](#).
- [Activate DataWorks](#).
- Create a DataWorks on the workflow. This example uses DataWorks simple mode. For more information, see [create a workflow](#).

Procedure

1. Prepare test data.
 - i. Log on to the [DataWorks console](#).
 - ii. Create a table and import data to the table. For more information, see [Create tables and import data](#). Execute the following statement to create a table named pytable2:

```
CREATE TABLE `pytable2` (
  `id` string,
  `name` string,
  `f1` double,
  `f2` double,
  `f3` double,
  `f4` double
);
```

Import the data in the pytable2.txt file to the pytable2 table. The pytable2.txt file contains the following data:

```
0, name1, 1.0, NaN, 3.0, 4.0
1, name1, 2.0, NaN, NaN, 1.0
2, name1, 3.0, 4.0, 1.0, NaN
3, name1, NaN, 1.0, 2.0, 3.0
4, name1, 1.0, NaN, 3.0, 4.0
5, name1, 1.0, 2.0, 3.0, 4.0
6, name1, NaN, NaN, NaN, NaN
```

2. In the left-side navigation pane, click **Workspaces**.
3. On the page that appears, find the workspace that you want to manage and click **Data Analytics** in the Actions column.
4. On the DataStudio page, right-click a workflow and choose **New > MaxCompute > PyODPS 2**.
5. In the **New node** dialog box, specify the **Node name** parameter and click **Submit**.
6. On the configuration tab of the PyODPS 2 node, enter the code of the node in the code

editor. In this example, enter the following code:

```
df2 = DataFrame(o.get_table('pytable2'))
# Call the dropna method to delete the rows that contain NULL values.
print df2.dropna(subset=['f1','f2','f3','f4']).head()

# If you do not want to delete the rows that also contain non-NULL values, specify how='all' for the dropna method.
print df2.dropna(how='all', subset=['f1','f2','f3','f4']).head()
print df2.dropna(thresh=3, subset=['f1', 'f2', 'f3', 'f4']).head()

# Call the fillna method to replace NULL values with a specified constant or values in an existing column.
print df2.fillna(100, subset=['f1','f2','f3','f4']).head()

# Replace NULL values with the values in an existing column.
print df2.fillna(df2.f2, subset=['f1','f2','f3','f4']).head()

# Replace each NULL value with the value in the same column of the previous row.
print df2.fillna(method='bfill', subset=['f1', 'f2', 'f3', 'f4']).head()

# Replace each NULL value with the value in the same column of the following row.
print df2.fillna(method='ffill', subset=['f1', 'f2', 'f3', 'f4']).head()
```

7. Click the Run icon in the toolbar.



8. View the running result of the PyODPS 2 node on the Run Log tab.



In this example, the following information appears on the Run Log tab:

```
Sql compiled:
CREATE TABLE tmp_pyodps_d0c7d8c2_be38_4d48_b0eb_e89bae5bde01 LIFECYCLE 1 AS
SELECT *
FROM WB_BestPractice_dev.`pytable2` t1
WHERE (((IF(t1.`f1` IS NOT NULL, 1, 0) + IF(t1.`f2` IS NOT NULL, 1, 0)) + IF(t1.`f3` IS NOT NULL, 1, 0))
+ IF(t1.`f4` IS NOT NULL, 1, 0)) >= 4
Instance ID: 20191010071154980g2hic292

id  name  f1  f2  f3  f4
0  0  name1  1.0  NaN  3.0  4.0
1  1  name1  2.0  NaN  NaN  1.0
2  2  name1  3.0  4.0  1.0  NaN
```

```
3 3 name1 NaN 1.0 2.0 3.0
```

```
4 4 name1 1.0 NaN 3.0 4.0
```

```
5 5 name1 1.0 2.0 3.0 4.0
```

```
6 6 name1 NaN NaN NaN NaN
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_49b46768_f589_48f6_be8a_b7139f31f6f2 LIFECYCLE 1 AS
```

```
SELECT *
```

```
FROM WB_BestPractice_dev.`pytable2` t1
```

```
WHERE (((IF(t1.`f1` IS NOT NULL, 1, 0) + IF(t1.`f2` IS NOT NULL, 1, 0)) + IF(t1.`f3` IS NOT NULL, 1, 0)) + IF(t1.`f4` IS NOT NULL, 1, 0)) >= 1
```

Instance ID: 20191010071159759g0dk9592

```
id name f1 f2 f3 f4
```

```
0 0 name1 1.0 NaN 3.0 4.0
```

```
1 1 name1 2.0 NaN NaN 1.0
```

```
2 2 name1 3.0 4.0 1.0 NaN
```

```
3 3 name1 NaN 1.0 2.0 3.0
```

```
4 4 name1 1.0 NaN 3.0 4.0
```

```
5 5 name1 1.0 2.0 3.0 4.0
```

```
6 6 name1 NaN NaN NaN NaN
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_7f941800_1539_415b_9257_283eb8b893a6 LIFECYCLE 1 AS
```

```
SELECT *
```

```
FROM WB_BestPractice_dev.`pytable2` t1
```

```
WHERE (((IF(t1.`f1` IS NOT NULL, 1, 0) + IF(t1.`f2` IS NOT NULL, 1, 0)) + IF(t1.`f3` IS NOT NULL, 1, 0)) + IF(t1.`f4` IS NOT NULL, 1, 0)) >= 3
```

Instance ID: 20191010071204544giyswx7

```
0 0 name1 1.0 NaN 3.0 4.0
```

```
1 1 name1 2.0 NaN NaN 1.0
```

```
2 2 name1 3.0 4.0 1.0 NaN
```

```
3 3 name1 NaN 1.0 2.0 3.0
```

```
4 4 name1 1.0 NaN 3.0 4.0
```

```
5 5 name1 1.0 2.0 3.0 4.0
```

```
6 6 name1 NaN NaN NaN NaN
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_16d6ea6d_5195_4e4c_8346_644a395852f7 LIFECYCLE 1 AS
```

```
SELECT t1.`id`, t1.`name`, IF(t1.`f1` IS NULL, 100, t1.`f1`) AS `f1`, IF(t1.`f2` IS NULL, 100, t1.`f2`) AS `f2`, IF(t1.`f3` IS NULL, 100, t1.`f3`) AS `f3`, IF(t1.`f4` IS NULL, 100, t1.`f4`) AS `f4`
```

```
2 , IF(t1.t3 IS NULL, 100, t1.t3 ) AS t3 , IF(t1.t4 IS NULL, 100, t1.t4 ) AS t4
```

```
FROM WB_BestPractice_dev.`pytable2` t1
```

Instance ID: 20191010071209190gyl56292

```
id name f1 f2 f3 f4
0 0 name1 1.0 NaN 3.0 4.0
1 1 name1 2.0 NaN NaN 1.0
2 2 name1 3.0 4.0 1.0 NaN
3 3 name1 NaN 1.0 2.0 3.0
4 4 name1 1.0 NaN 3.0 4.0
5 5 name1 1.0 2.0 3.0 4.0
6 6 name1 NaN NaN NaN NaN
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_40755ebd_2d2a_482e_b360_3f3da0d5422c LIFECYCLE 1 AS
SELECT t1.`id`, t1.`name`, IF(t1.`f1` IS NULL, t1.`f2`, t1.`f1`) AS `f1`, IF(t1.`f2` IS NULL, t1.`f2`, t1.`f2`
) AS `f2`, IF(t1.`f3` IS NULL, t1.`f2`, t1.`f3`) AS `f3`, IF(t1.`f4` IS NULL, t1.`f2`, t1.`f4`) AS `f4`
FROM WB_BestPractice_dev.`pytable2` t1
```

Instance ID: 20191010071213970gbp66792

```
id name f1 f2 f3 f4
0 0 name1 1.0 NaN 3.0 4.0
1 1 name1 2.0 NaN NaN 1.0
2 2 name1 3.0 4.0 1.0 NaN
3 3 name1 NaN 1.0 2.0 3.0
4 4 name1 1.0 NaN 3.0 4.0
5 5 name1 1.0 2.0 3.0 4.0
6 6 name1 NaN NaN NaN NaN
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_d39fcce1_d8a9_4cc2_8aff_2ed1e9c6bb1b LIFECYCLE 1 AS
SELECT pyodps_udf_1570691538_d9441c59_c666_4a5d_8154_67d8bc8c24ad(t1.`id`, t1.`name`, t1.`f1`,
t1.`f2`, t1.`f3`, t1.`f4`) AS (`id`, `name`, `f1`, `f2`, `f3`, `f4`)
FROM WB_BestPractice_dev.`pytable2` t1
```

Instance ID: 20191010071219627goqv9292

```
id name f1 f2 f3 f4
0 0 name1 1.0 3.0 3.0 4.0
1 1 name1 2.0 1.0 1.0 1.0
2 2 name1 3.0 4.0 1.0 NaN
```

```
3 3 name1 1.0 1.0 2.0 3.0
```

```
4 4 name1 1.0 3.0 3.0 4.0
```

```
5 5 name1 1.0 2.0 3.0 4.0
```

```
6 6 name1 NaN NaN NaN NaN
```

Sql compiled:

```
CREATE TABLE tmp_pyodps_3f190cf0_f9fb_4e06_a942_ab31c0241cd3 LIFECYCLE 1 AS
```

```
SELECT pyodps_udf_1570691566_0330848b_82d3_411c_88e1_cbbcc6adb9c1(t1.`id`, t1.`name`, t1.`f1`,  
t1.`f2`, t1.`f3`, t1.`f4`) AS (`id`, `name`, `f1`, `f2`, `f3`, `f4`)
```

```
FROM WB_BestPractice_dev.`pytable2` t1
```

Instance ID: 20191010071247729gt776792

```
id name f1 f2 f3 f4
```

```
0 0 name1 1.0 1.0 3.0 4.0
```

```
1 1 name1 2.0 2.0 2.0 1.0
```

```
2 2 name1 3.0 4.0 1.0 1.0
```

```
3 3 name1 NaN 1.0 2.0 3.0
```

```
4 4 name1 1.0 1.0 3.0 4.0
```

```
5 5 name1 1.0 2.0 3.0 4.0
```

```
6 6 name1 NaN NaN NaN NaN
```

7. Spark

7.1. Spark on MaxCompute overview

This topic provides an overview of Spark on MaxCompute. It is an open-source framework that functions on the service level to support data processing and analysis operations. Equipped with unified computing resources and data set permissions, Spark on MaxCompute allows you to submit and run jobs while using your preferred development methods.

Features

- Supports different versions of native Spark jobs.

MaxCompute is compatible with the APIs of all native Spark versions that run in MaxCompute. Different versions of Spark can run in MaxCompute at the same time. Spark on MaxCompute provides you with native Spark Web UIs.

- Runs in unified computing resources.

Similar to MaxCompute SQL and MaxCompute MapReduce, Spark on MaxCompute runs in the unified computing resources activated for MaxCompute projects.

- Supports unified data and permission management.

Spark on MaxCompute complies with the permissions you set for MaxCompute projects, allowing you to query data without any additional permission modifications required.

- Provides the same experience as open-source systems.

Spark on MaxCompute provides the same user experience as open-source systems, both in terms of an open-source application UI and online interactions. Specifically, it supports native, open-source, and real-time UIs that are essential for debugging open-source applications, while also provides the historical log query function.

Architecture

The Spark on MaxCompute architecture solution allows native Spark engines to run in MaxCompute. The following figure shows this architecture.

□

The left half of the diagram shows the architecture of a native Spark engine, and the right half shows the architecture of Spark on MaxCompute. This second architecture runs on the Cupid platform developed by Alibaba Cloud. This platform also supports computing frameworks such as native Spark engines.

Supported services

MaxCompute Spark supports the following services:

- All Java and Scala offline jobs such as GraphX, MLlib, RDD, Spark SQL, and PySpark
- Read and write operations on MaxCompute tables
- Unstructured storage in OSS

MaxCompute Spark will support the following services in its later versions:

- Read and write operations on services such as RDS, Redis, and ECS in VPCs
- Streaming services

- Interactive services such as Spark shell, Spark SQL shell, and PySpark shell

7.2. Set up a Spark on MaxCompute development environment

This topic describes how to set up a Spark on MaxCompute development environment. To do so, you need to download the Spark on MaxCompute client, set environment variables, configure the *spark-defaults.conf* file, and configure dependencies.

Download the Spark on MaxCompute client

The Spark on MaxCompute software packages are interoperable with the authentication function of MaxCompute. This allows Spark on MaxCompute to serve as a client that submits jobs with the *spark-submit* script that is encrypted. The following two Spark on MaxCompute software packages are provided to meet different needs:

- **spark-1.6.3**: Develops Spark1.x applications.
- **spark-2.3.0**: Develops Spark2.x applications.

Set environment variables

- Set the *JAVA_HOME* environment variable as follows:

```
# We recommend that you use JDK 1.8 or later.
export JAVA_HOME=/path/to/jdk
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
```

- Set the *SPARK_HOME* environment variable as follows:

```
export SPARK_HOME=/path/to/spark_extracted_package
export PATH=$SPARK_HOME/bin:$PATH
```

- If you use PySpark, install Python 2.7 and set the *PATH* environment variable as follows:

```
export PATH=/path/to/python/bin/:$PATH
```

Configure the *spark-defaults.conf* file

You can use the *spark-defaults.conf.template* file in the *\$SPARK_HOME/conf* directory as a template to prepare your own *spark-defaults.conf* file. However, before you submit Spark on MaxCompute jobs to MaxCompute, you must set the required MaxCompute account and region information in your *spark-defaults.conf* file.

In the *spark-defaults.conf* file, you can retain the default settings and enter the MaxCompute account information as follows:


```
# Set the MaxCompute project and account information:
spark.hadoop.odps.project.name =
spark.hadoop.odps.access.id =
spark.hadoop.odps.access.key =

# Configure the endpoint through which the Spark on MaxCompute client accesses MaxCompute projects (this endpoint varies depending on the network conditions and region):
spark.hadoop.odps.end.point = http://service.cn.maxcompute.aliyun.com/api
# Configure the endpoint that runs Spark on MaxCompute (this endpoint runs in the MaxCompute VPC in your region):
spark.hadoop.odps.runtime.end.point = http://service.cn.maxcompute.aliyun-inc.com/api

# Retain the following default settings:
spark.sql.catalogImplementation=odps
spark.hadoop.odps.task.major.version = cupid_v2
spark.hadoop.odps.cupid.container.image.enable = true
spark.hadoop.odps.cupid.container.vm.engine.type = hyper
```

Configure dependencies

- Configure the dependencies for Spark on MaxCompute jobs to access MaxCompute tables.

Spark on MaxCompute jobs use the `odps-spark-datasource` module to access MaxCompute tables. The Maven coordinates of this module are as follows:

```
<!-- Spark-2.x uses the following module:-->
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-spark-datasource_2.11</artifactId>
  <version>3.3.3-public</version>
</dependency>

<!-- Spark-1.x uses the following module:-->
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-spark-datasource_2.10</artifactId>
  <version>3.3.3-public</version>
</dependency>
```

- Configure the dependencies for Spark on MaxCompute jobs to access OSS.

If Spark on MaxCompute jobs need to access OSS, add the following dependencies:

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>hadoop-fs-oss</artifactId>
  <version>3.3.3-public</version>
</dependency>
```

7.3. Running modes

MaxCompute Spark supports three running modes: Local, Cluster, and DataWorks.

Local mode Cluster mode DataWorks mode

Local mode

The Local mode is used to facilitate code debugging for applications. In Local mode, you can use MaxCompute Spark the same way as native Spark in the community. Additionally, you can use Tunnel to read data from and write data to MaxCompute tables. In this mode, you can use either an IDE or the command line to run MaxCompute Spark. If you use this mode, you must add the `spark.master=local[N]` configuration. N indicates the CPU resources required to implement this mode. To use Tunnel to read data from and write data to tables in Local mode, you must add the Tunnel configuration item to *Spark-defaults.conf*. Enter the endpoint based on the region and network environment where the MaxCompute project is located. For more information about how to obtain the endpoint, see [Configure endpoints](#). The following code provides an example on how to use the command line to run MaxCompute Spark in this mode:

```
1.bin/spark-submit --master local[4] \
--class com.aliyun.odps.spark.examples.SparkPi \
${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/target/spark-examples_2.11-version-shaded.jar
```

Cluster mode

In Cluster mode, you must specify the Main method as the entry point of a custom application. A Spark job ends when Main succeeds or fails. This mode is suitable for offline jobs. You can use MaxCompute Spark in this mode together with DataWorks to schedule jobs. The following code provides an example on how to use the command line to run MaxCompute Spark in this mode:

```
1.bin/spark-submit --master yarn-cluster \
-class SparkPi \
${ProjectRoot}/spark/spark-2.x/spark-examples/target/spark-examples_2.11-version-shaded.jar
```

DataWorks mode

You can run offline jobs of MaxCompute Spark (in Cluster mode) in DataWorks to integrate and schedule the other types of nodes.

 **Note** DataWorks supports the Spark node in the following regions: China (Hangzhou), China (Beijing), China (Shanghai), China (Shenzhen), China (Hong Kong), US (Silicon Valley), Germany (Frankfurt), India (Mumbai), and Singapore.

To use MaxCompute Spark in this mode, follow these steps:

1. Upload the resources in the DataWorks business flow and click **Submit**.

After the resources are uploaded, the following figure appears.

2. In the created business flow, select **ODPS Spark** from **Data Analytics**.
3. Double-click the Spark node and define the Spark job. Select a Spark version, a development language, and a resource file. The resource file is the file uploaded and published in the business flow. You can specify configuration items, such as the number of executors and memory size, for the job to be submitted. You also need to set `spark.hadoop.odps.cupid.webproxy.endpoint` to the endpoint of the region where the project is located, for example, `http://service.cn.maxcompute.aliyun-inc.com/api`.
4. Run the Spark node to view the task operational log and obtain the URLs of LogView and JobView from the log for further analysis and diagnosis.

After you have defined the Spark job, orchestrate and schedule services of different types in the business flow if required.

7.4. Java and Scala development examples

7.4.1. Develop a Spark on MaxCompute application by using Java or Scala

This topic describes how to develop a Spark on MaxCompute application by using Java or Scala.

Download an example project

You can run the following commands to download an example project:

```
git clone git@github.com:aliyun/aliyun-cupid-sdk.git
cd aliyun-cupid-sdk
git checkout 3.3.3-public
# Download an example project for Spark-2.x.
cd spark/spark-2.x/spark-examples
# Download an example project for Spark-1.x.
cd spark/spark-1.x/spark-examples
# Package data to create a shaded JAR package in the target directory.
mvn clean package
```

Configure dependencies for Spark-1.x

If you want to submit your Spark-1.x application by using the Spark on MaxCompute client, you must add the following dependencies to the *pom.xml* file:

```
<properties>
  <spark.version>1.6.3</spark.version>
  <cupid.sdk.version>3.3.3-public</cupid.sdk.version>
  <scala.version>2.10.4</scala.version>
  <scala.binary.version>2.10</scala.binary.version>
</properties>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_${scala.binary.version}</artifactId>
  <version>${spark.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_${scala.binary.version}</artifactId>
  <version>${spark.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-mllib_${scala.binary.version}</artifactId>
  <version>${spark.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-streaming_${scala.binary.version}</artifactId>
  <version>${spark.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>cupid-sdk</artifactId>
  <version>${cupid.sdk.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>hadoop-fs-oss</artifactId>
```

```

    <artifactId>nadoop-ts-oss</artifactId>
    <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-spark-datasource_${scala.binary.version}</artifactId>
    <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId>
    <version>${scala.version}</version>
</dependency>
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-actors</artifactId>
    <version>${scala.version}</version>
</dependency>

```

 **Note** You need to set the scope parameter as follows:

- Set it to *provided* for all packages that are released in the Apache Spark community, such as spark-core and spark-sql.
- Set it to *compile* for the odps-spark-datasource module.

Develop a Spark-1.x application

- Develop the WordCount application.

For this application, you will need to download [aliyun-cupid-sdk](#).

To submit the code, follow these steps:

- Build the aliyun-cupid-sdk module.
- Configure the *spark.defaults.conf* file.
- Run the following script:

```

bin/spark-submit --master yarn-cluster --class \
com.aliyun.odps.spark.examples.WordCount \
${path to aliyun-cupid-sdk}/spark/spark-1.x/spark-examples/target/spark-examples_2.10-versi
on-shaded.jar

```

- Develop the Spark SQL application on MaxCompute tables.

For this application, you will need to download [aliyun-cupid-sdk](#).

To submit the code, follow these steps:

- i. Build the aliyun-cupid-sdk module.
- ii. Configure the *spark.defaults.conf* file.
- iii. Run the following script:

```
bin/spark-submit --master yarn-cluster --class \  
com.aliyun.odps.spark.examples.sparksql.SparkSQL \  
${path to aliyun-cupid-sdk}/spark/spark-1.x/spark-examples/target/spark-examples_2.10-versi  
on-shaded.jar
```

 Note

- If the "Table Not Found" error is returned, the table you specify in the code cannot be found in the MaxCompute project.
- You can develop a Spark SQL application for the target table with reference to various APIs in the code.

- Develop the GraphX PageRank application.

For this application, you will need to download [aliyun-cupid-sdk](#).

To submit the code, follow these steps:

- i. Build the aliyun-cupid-sdk module.
- ii. Configure the *spark.defaults.conf* file.
- iii. Run the following script:

```
bin/spark-submit --master yarn-cluster --class \  
com.aliyun.odps.spark.examples.graphx.PageRank \  
${path to aliyun-cupid-sdk}/spark/spark-1.x/spark-examples/target/spark-examples_2.10-versi  
on-shaded.jar
```

- Develop the MLlib Kmeans-ON-OSS application.

For this application, you will need to download [aliyun-cupid-sdk](#).

 **Note** Before you submit the code, make sure that you enter the following OSS account information in the code:

```
conf.set("spark.hadoop.fs.oss.accessKeyId", "****")  
conf.set("spark.hadoop.fs.oss.accessKeySecret", "****")  
conf.set("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.aliyuncs.com")
```

To submit the code, follow these steps:

- i. Build the aliyun-cupid-sdk module.
- ii. Configure the *spark.defaults.conf* file.
- iii. Run the following script:

```
bin/spark-submit --master yarn-cluster --class \
com.aliyun.odps.spark.examples.mllib.KmeansModelSaveToOss \
${path to aliyun-cupid-sdk}/spark/spark-1.x/spark-examples/target/spark-examples_2.10-versi
on-shaded.jar
```

- Develop the OSS UnstructuredData application.

For this application, you will need to download [aliyun-cupid-sdk](#).

 **Note** Before you submit the code, make sure that you enter the following OSS account information in the code:

```
conf.set("spark.hadoop.fs.oss.accessKeyId", "****")
conf.set("spark.hadoop.fs.oss.accessKeySecret", "****")
conf.set("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.aliyuncs.com")
```

To submit the code, follow these steps:

- Build the aliyun-cupid-sdk module.
- Configure the *spark.defaults.conf* file.
- Run the following script:

```
bin/spark-submit --master yarn-cluster --class \
com.aliyun.odps.spark.examples.oss.SparkUnstructuredDataCompute \
${path to aliyun-cupid-sdk}/spark/spark-1.x/spark-examples/target/spark-examples_2.10-versi
on-shaded.jar
```

Configure dependencies for Spark-2.x

If you want to submit your Spark-2.x application by using the Spark on MaxCompute client, you must add the following dependencies to the *pom.xml* file:

```
<properties>
  <spark.version>2.3.0</spark.version>
  <cupid.sdk.version>3.3.3-public</cupid.sdk.version>
  <scala.version>2.11.8</scala.version>
  <scala.binary.version>2.11</scala.binary.version>
</properties>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_${scala.binary.version}</artifactId>
  <version>${spark.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
```

```
<groupId>org.apache.spark</groupId>
<artifactId>spark-sql_${scala.binary.version}</artifactId>
<version>${spark.version}</version>
<scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-mllib_${scala.binary.version}</artifactId>
  <version>${spark.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-streaming_${scala.binary.version}</artifactId>
  <version>${spark.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>cupid-sdk</artifactId>
  <version>${cupid.sdk.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>hadoop-fs-oss</artifactId>
  <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-spark-datasource_${scala.binary.version}</artifactId>
  <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
  <groupId>org.scala-lang</groupId>
  <artifactId>scala-library</artifactId>
  <version>${scala.version}</version>
</dependency>
<dependency>
  <groupId>org.scala-lang</groupId>
  <artifactId>scala-actors</artifactId>
```



```
<version>${scala.version}</version>
</dependency>
```

 **Note** You need to set the scope parameter as follows:

- Set it to *provided* for all packages that are released in the Apache Spark community, such as spark-core and spark-sql.
- Set it to *compile* for the odps-spark-datasource module.

Develop a Spark-2.x application

- Develop the WordCount application.

For this application, you will need to download [aliyun-cupid-sdk](#).

To submit the code, follow these steps:

- Build the aliyun-cupid-sdk module.
- Configure the *spark.defaults.conf* file.
- Run the following script:

```
bin/spark-submit --master yarn-cluster --class \
com.aliyun.odps.spark.examples.WordCount \
${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/target/spark-examples_2.11-versi
on-shaded.jar
```

- Develop the Spark SQL application on MaxCompute tables.

For this application, you will need to download [aliyun-cupid-sdk](#).

 **Note**

- If the "Table Not Found" error is returned, the table you specify in the code cannot be found in the MaxCompute project.
- You can develop a Spark SQL application for the target table with reference to various APIs in the code.

To submit the code, follow these steps:

- Build the aliyun-cupid-sdk module.
- Configure the *spark.defaults.conf* file.
- Run the following script:

```
bin/spark-submit --master yarn-cluster --class \
com.aliyun.odps.spark.examples.sparksql.SparkSQL \
${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/target/spark-examples_2.11-versi
on-shaded.jar
```

- Develop the GraphX PageRank application.

For this application, you will need to download [aliyun-cupid-sdk](#).

To submit the code, follow these steps:

- i. Build the aliyun-cupid-sdk module.
- ii. Configure the *spark.defaults.conf* file.
- iii. Run the following script:

```
bin/spark-submit --master yarn-cluster --class \
com.aliyun.odps.spark.examples.graphx.PageRank \
${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/target/spark-examples_2.11-versi
on-shaded.jar
```

- Develop the MLlib Kmeans-ON-OSS application.

For this application, you will need to download [aliyun-cupid-sdk](#).

 **Note** Before you submit the code, make sure that you enter the following OSS account information in the code:

```
val spark = SparkSession
  .builder()
  .config("spark.hadoop.fs.oss.accessKeyId", "****")
  .config("spark.hadoop.fs.oss.accessKeySecret", "****")
  .config("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.aliyuncs.com")
  .appName("KmeansModelSaveToOss")
  .getOrCreate()
```

To submit the code, follow these steps:

- i. Build the aliyun-cupid-sdk module.
- ii. Configure the *spark.defaults.conf* file.
- iii. Run the following script:

```
bin/spark-submit --master yarn-cluster --class \
com.aliyun.odps.spark.examples.mllib.KmeansModelSaveToOss \
${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/target/spark-examples_2.11-versi
on-shaded.jar
```

- Develop the OSS UnstructuredData application.

For this application, you will need to download [aliyun-cupid-sdk](#).

 **Note** Before you submit the code, make sure that you enter the following OSS account information in the code:

```
val spark = SparkSession
    .builder()
    .config("spark.hadoop.fs.oss.accessKeyId", "****")
    .config("spark.hadoop.fs.oss.accessKeySecret", "****")
    .config("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.aliyuncs.com")
    .appName("SparkUnstructuredDataCompute")
    .getOrCreate()
```

To submit the code, follow these steps:

- i. Build the aliyun-cupid-sdk module.
- ii. Configure the *spark.defaults.conf* file.
- iii. Run the following script:

```
bin/spark-submit --master yarn-cluster --class \
com.aliyun.odps.spark.examples.oss.SparkUnstructuredDataCompute \
${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/target/spark-examples_2.11-version-shaded.jar
```

7.4.2. Spark-2.x examples

This topic shows how to configure Spark-2.x dependencies, including some examples.

Configure dependencies for Spark-2.x

If you want to submit your Spark-2.x application by using the Spark on MaxCompute client, you must add the following dependencies in the *pom.xml* file. For more information about *pom.xml*, see [pom.xml](#).

```
<properties>
  <spark.version>2.3.0</spark.version>
  <cupid.sdk.version>3.3.8-public</cupid.sdk.version>
  <scala.version>2.11.8</scala.version>
  <scala.binary.version>2.11</scala.binary.version>
</properties>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_${scala.binary.version}</artifactId>
  <version>${spark.version}</version>
  <scope>provided</scope>
</dependency>
```

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_${scala.binary.version}</artifactId>
  <version>${spring-version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-mllib_${scala.binary.version}</artifactId>
  <version>${spark.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-streaming_${scala.binary.version}</artifactId>
  <version>${spark.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>cupid-sdk</artifactId>
  <version>${cupid.sdk.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>hadoop-fs-oss</artifactId>
  <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-spark-datasource_${scala.binary.version}</artifactId>
  <version>${cupid.sdk.version}</version>
</dependency>
<dependency>
  <groupId>org.scala-lang</groupId>
  <artifactId>scala-library</artifactId>
  <version>${scala.version}</version>
</dependency>
<dependency>
  <groupId>org.scala-lang</groupId>
```

```
<artifactId>scala-actors</artifactId>
<version>${scala.version}</version>
</dependency>
```

In the preceding code, set the scope parameter as follows:

- Set it to provided for all packages that are released in the Apache Spark community, such as spark-core and spark-sql.
- Set it to compile for the odps-spark-datasource module.

WordCount example

- [Detailed code](#)
- How to submit

```
cd /path/to/MaxCompute-Spark/spark-2.x
mvn clean package

# For the configuration of the environment variables in the spark-defaults.conf file, see Set up a Sp
ark on MaxCompute development environment.
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class \
    com.aliyun.odps.spark.examples.WordCount \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

Example of Spark-SQL on MaxCompute Table

- [Detailed code](#)
- How to submit

```
# If the table you specify in the code cannot be found in the MaxCompute project, a "Table Not Foun
d" error will be returned.
# You can develop a Spark SQL application for the target table with reference to various APIs in the
code.
Step 1. build aliyun-cupid-sdk
Step 2. properly set spark.defaults.conf
Step 3. bin/spark-submit --master yarn-cluster --class \
    com.aliyun.odps.spark.examples.sparksql.SparkSQL \
    ${path to aliyun-cupid-sdk}/spark/spark-2.x/spark-examples/target/spark-examples_2.11-versio
n-shaded.jar
```

GraphX PageRank example

- [Detailed code](#)
- How to submit

```
cd /path/to/MaxCompute-Spark/spark-2.x
mvn clean package
# For more information about the configuration of the environment variables in the spark-defaults.conf file, see Set up a Spark on MaxCompute development environment.
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class \
    com.aliyun.odps.spark.examples.graphx.PageRank \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

Mllib Kmeans-ON-OSS examples

For more information about how to configure `spark.hadoop.fs.oss.sts.token.roleArn` and `spark.hadoop.fs.oss.endpoint`, see [OSS access notes](#).

- [Detailed code](#)
- How to submit

```
# Edit code
val modelOssDir = "oss: // bucket/kmeans-model" // Enter the specific OSS Bucket path.
val spark = SparkSession
    .builder()
    .config("spark.hadoop.fs.oss.credentials.provider", "org.apache.hadoop.fs.aliyun.oss.AliyunStsTokenCredentialsProvider")
    .config("spark.hadoop.fs.oss.sts.token.roleArn", "acs:ram::****:role/aliyunodpsdefaultrole")
    .config("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.aliyuncs.com")
    .appName("KmeansModelSaveToOss")
    .getOrCreate()

cd /path/to/MaxCompute-Spark/spark-2.x
mvn clean package
# For more information about configuration of the environment variables in the spark-defaults.conf file, see Set up a Spark on MaxCompute development environment.
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class \
    com.aliyun.odps.spark.examples.mllib.KmeansModelSaveToOss
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

OSS UnstructuredData example

For more information about how to configure `spark.hadoop.fs.oss.sts.token.roleArn` and `spark.hadoop.fs.oss.endpoint`, see [OSS access notes](#).

- [Detailed code](#)
- How to submit

```
# Edit code
val pathIn = "oss: // bucket/inputdata/" // Enter the specific OSS Bucket path.
val spark = SparkSession
    .builder()
    .config("spark.hadoop.fs.oss.credentials.provider", "org.apache.hadoop.fs.aliyun.oss.AliyunStsTokenCredentialsProvider")
    .config("spark.hadoop.fs.oss.sts.token.roleArn", "acs:ram::****:role/aliyunodpsdefaultrole")
    .config("spark.hadoop.fs.oss.endpoint", "oss-cn-hangzhou-zmf.aliyuncs.com")
    .appName("SparkUnstructuredDataCompute")
    .getOrCreate()

cd /path/to/MaxCompute-Spark/spark-2.x
mvn clean package
# For more information about the configuration of the environment variables in the spark-defaults.conf file, see Set up a Spark on MaxCompute development environment.
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class \
    com.aliyun.odps.spark.examples.oss.SparkUnstructuredDataCompute \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

MaxCompute table I/O example

- [Detailed code](#)
- How to submit

```
cd /path/to/MaxCompute-Spark/spark-2.x
mvn clean package
# For more information about the configuration of the environment variables in the spark-defaults.conf file, see Set up a Spark on MaxCompute development environment.
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class \
    com.aliyun.odps.spark.examples.sparksql.SparkSQL \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

Example of PySpark I/O by MaxCompute table

- [Detailed code](#)
- How to submit

```
# For more information about the configuration of the environment variables in the spark-defaults.conf file, see Set up a Spark on MaxCompute development environment.
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --jars /path/to/odps-spark-datasource_2.11-3.3.8-public.jar \
/path/to/MaxCompute-Spark/spark-2.x/src/main/python/spark_sql.py
```

PySpark writing to OSS example

- [Detailed code](#)
- How to submit

```
# For more information about the configuration of the environment variables in the spark-defaults.conf file, see Set up a Spark on MaxCompute development environment.
# For more information about OSS configuration, see OSS access notes.

cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --jars /path/to/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar \
/path/to/MaxCompute-Spark/spark-2.x/src/main/python/spark_oss.py
# Compile Spark-2.x to get spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar.
```

Example of supporting Spark Streaming Loghub

- [Detailed code](#)
- How to submit

```
# For more information about the configuration of the environment variables in the spark-defaults.conf file, see Set up a Spark on MaxCompute development environment.
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class \
com.aliyun.odps.spark.examples.streaming.loghub.LogHubStreamingDemo \
/path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

Example of supporting Spark Streaming Datahub

- [Detailed code](#)
- How to submit


```
# For more information about the configuration of the environment variables in the spark-defaults.conf file, see Set up a Spark on MaxCompute development environment.
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class \
    com.aliyun.odps.spark.examples.streaming.datahub.DataHubStreamingDemo \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

Example of supporting Spark Streaming Kafka

- [Detailed code](#)
- How to submit

```
# For more information about the configuration of the environment variables in the spark-defaults.conf file, see Set up a Spark on MaxCompute development environment.
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class \
    com.aliyun.odps.spark.examples.streaming.kafka.KafkaStreamingDemo \
    /path/to/MaxCompute-Spark/spark-2.x/target/spark-examples_2.11-1.0.0-SNAPSHOT-shaded.jar
```

 **Note** For more information, see [MaxCompute-Spark](#).

7.5. Develop a Spark on MaxCompute application by using PySpark

This topic describes how to develop a Spark on MaxCompute application by using PySpark.

If you want to access MaxCompute tables, you must compile a package for the odps-spark-datasource package. For more information, see [Set up a Spark on MaxCompute development environment](#).

Develop a Spark SQL application in Spark 1.6

1. Compile the following code:

```
from pyspark import SparkContext, SparkConf
from pyspark.sql import OdpsContext
if __name__ == '__main__':
    conf = SparkConf().setAppName("odps_pyspark")
    sc = SparkContext(conf=conf)
    sql_context = OdpsContext(sc)
    sql_context.sql("DROP TABLE IF EXISTS spark_sql_test_table")
    sql_context.sql("CREATE TABLE spark_sql_test_table(name STRING, num BIGINT)")
    sql_context.sql("INSERT INTO TABLE spark_sql_test_table SELECT 'abc', 100000")
    sql_context.sql("SELECT * FROM spark_sql_test_table").show()
    sql_context.sql("SELECT COUNT(*) FROM spark_sql_test_table").show()
```

2. Run the following command to submit the code:

```
./bin/spark-submit \
--jars cupid/odps-spark-datasource_xxx.jar \
example.py
```

Develop a Spark SQL application in Spark 2.3

1. Compile the following code:

```
from pyspark.sql import SparkSession
if __name__ == '__main__':
    spark = SparkSession.builder.appName("spark sql").getOrCreate()
    spark.sql("DROP TABLE IF EXISTS spark_sql_test_table")
    spark.sql("CREATE TABLE spark_sql_test_table(name STRING, num BIGINT)")
    spark.sql("INSERT INTO spark_sql_test_table SELECT 'abc', 100000")
    spark.sql("SELECT * FROM spark_sql_test_table").show()
    spark.sql("SELECT COUNT(*) FROM spark_sql_test_table").show()
```

2. Run one of the following commands based on which mode you use:

- In cluster mode, run the following command to submit the code:

```
spark-submit --master yarn-cluster \
--jars cupid/odps-spark-datasource_xxx.jar \
example.py
```

- In local mode, run the following command to submit the code:

```
cd $SPARK_HOME
./bin/spark-submit --master local[4] \
--driver-class-path cupid/odps-spark-datasource_xxx.jar \
/path/to/odps-spark-examples/spark-examples/src/main/python/spark_sql.py
```

Required packages

A Python library cannot be directly installed in the MaxCompute cluster. When PySpark depends on a Python library, plugin, or project, you need to package this Python library, plugin, or project on your computer and then upload the package by running the `spark-submit` script. The Python version that you use to package the Python library, plugin, or project must be the same as the Python version in which your Spark on MaxCompute jobs run. You can choose one of the following two packaging formats: egg and Python.

- Compile egg packages on your computer.

For example, if MLlib requires the NumPy and Setuptools plugins, you need to compile egg packages for them and then upload the packages by running the `--py-files` script. Detailed steps are as follows:

 **Note** Spark on MaxCompute jobs run in Python 2.7 or Python 3.5. Therefore, you must package the plugins by using Python 2.7 or Python 3.5 on your computer.

- i. To compile egg packages for the NumPy and Setuptools plugins, follow these steps:
 - a. Download the NumPy and Setuptools software packages found in [Find, install and publish Python packages with the Python Package Index](#).
 - b. Enter the source code path of the Setuptools plugin and run the `python setup.py bdist_egg` script. An egg file is generated in the `dist` directory.
 - c. Enter the source code path of the NumPy plugin and run the `python setup.py bdist_egg` script. An egg file is generated in the `dist` directory.
- ii. Run the following command to submit your Spark on MaxCompute jobs:

```
cd $SPARK_HOME
./bin/spark-submit --master yarn-cluster \
--jars cupid/odps-spark-datasource_2.11-3.3.2-hotfix1.jar \
--py-files /path/to/numpy-1.7.1-py2.7-linux-x85_64.egg,/path/to/setuptools-33.1.1-py2.7.egg \
app.py
```

- Compile Python packages on your computer.

If the Spark on MaxCompute application depends on a large number of plugins or any of these plugins contain files such as `.so` files that cannot be imported by the `zipimport` module, you need to download all the plugins, compile Python packages for the plugins, and then upload the packages. Detailed steps are as follows:

- i. Add the following configuration information:

```
spark.pyspark.python=./public.python-2.7-ucs4.zip/python-2.7-ucs4/bin/python2.7
```

- ii. Run the following command to submit your Spark on MaxCompute jobs:

```
cd $SPARK_HOME
./bin/spark-submit --master yarn-cluster \
--jars cupid/odps-spark-datasource_2.11-3.3.2-hotfix1.jar \
--archives ./python-2.7-ucs4.zip app.py
```

If you do not want to submit your Spark on MaxCompute jobs by running the `--archives` script, you can submit them as public resources.

- i. Add the following configuration information:

```
spark.hadoop.odps.cupid.resources=public.python-2.7-ucs4.zip
spark.pyspark.python=./public.python-2.7-ucs4.zip/python-2.7-ucs4/bin/python2.7
```

- ii. Run the following command to submit your Spark on MaxCompute jobs:

```
cd $SPARK_HOME
./bin/spark-submit --master yarn-cluster \
--jars cupid/odps-spark-datasource_2.11-3.3.2-hotfix1.jar app.py
```

If the Spark on MaxCompute application also depends on other plugins, you can package them with reference to the following script:

```
work_root=`dirname $0`
work_root=`cd ${work_root}; pwd`
# Step 1 compile python
# 1.1 python source code
cd ${work_root}
if [ ! -f Python-2.7.13.tgz ]; then
    wget https://www.python.org/ftp/python/2.7.13/Python-2.7.13.tgz
fi
# 1.2 configure && make && make install
if [ ! -d ${work_root}/Python-2.7.13 ]; then
    cd ${work_root}
    tar xf ${work_root}/Python-2.7.13.tgz
fi
if [ -d ${work_root}/python-2.7-ucs4 ]; then
    rm -rf ${work_root}/python-2.7-ucs4
fi
cd ${work_root}/Python-2.7.13
./configure --prefix=${work_root}/python-2.7-ucs4 --enable-unicode=ucs4
sed -i 's/#.*zlib zlibmodule.c/zlib zlibmodule.c/g' Modules/Setup
make -j20
make install
# 1.3 install pip
cd ${work_root}
if [ ! -f get-pip.py ]; then
    curl -s https://bootstrap.pypa.io/get-pip.py -o ${work_root}/get-pip.py
fi
${work_root}/python-2.7-ucs4/bin/python ${work_root}/get-pip.py
# 1.4 install numpy
${work_root}/python-2.7-ucs4/bin/pip install numpy
# 1.6 make python zip
if [ -f ${work_root}/python-2.7-ucs4.zip ]; then
    rm -rf ${work_root}/python-2.7-ucs4.zip
fi
cd ${work_root}
zip -r ${work_root}/python-2.7-ucs4.zip python-2.7-ucs4
```

7.6. Diagnose a Spark on MaxCompute job

This topic describes how to diagnose a Spark on MaxCompute job based on the job log. You can use LogView or the Spark Web UI to check whether a Spark on MaxCompute job is submitted or run.

Background information

When you submit a Spark on MaxCompute job by running the `spark-submit` script, MaxCompute creates an instance and adds instance information to the LogView log.

Specifically, you can submit a Spark on MaxCompute job through running the following `spark-submit` script:

```
cd $SPARK_HOME
bin/spark-submit --master yarn-cluster --class SparkPi /tmp/spark-2.x-demo/target/Alispark-2.x-quickstart-1.0-SNAPSHOT-shaded.jar
```

After the job is submitted, MaxCompute creates an instance and adds the instance information to the LogView log as follows:

```
19/01/05 20:36:47 INFO YarnClientImplUtil: logview url: http://logview.odps.aliyun.com/logview/?h=http://service.cn.maxcompute.aliyun.com/api&p=qn_beijing&i=xxx&token=xxx
```

If the job is submitted, the logged information includes but is not limited to the following:

```
19/01/05 20:37:34 INFO Client:
```

```
client token: N/A
```

```
diagnostics: N/A
```

```
ApplicationMaster host: 11.220.xxx.xxx
```

```
ApplicationMaster RPC port: 30002
```

```
queue: queue
```

```
start time: 1546691807945
```

```
final status: SUCCEEDED
```

```
tracking URL: http://jobview.odps.aliyun.com/proxyview/jobview/?h=http://service.cn.maxcompute.aliyun-inc.com/api&p=project_name&i=xxx&t=spark&id=application_xxx&metaname=xxx&token=xxx
```

 **Note** When you run a Spark on MaxCompute task in DataWorks, a similar log is created.

Diagnose a Spark on MaxCompute job by using LogView

1. Open a browser. Then use LogView to view the basic information about your Spark on MaxCompute job, a cupid-type task.
 -
2. On the **Fuxi Jobstab** in the upper pane, find the task named **master-0** in the **TaskName** column. Then, in the lower pane, click the **ALL** tab.
 -
3. On the **TempRoot** tab, find the log you want to view, and click the icon in the **StdOut** column. Then you can view the log details generated by SparkPi.
 -

Diagnose a Spark on MaxCompute job by using the Spark Web UI

If the log for a Spark on MaxCompute job contains a tracking URL, the job is submitted to the MaxCompute cluster. Both the Spark Web UI and the History Server use this tracking URL.

1. Open a browser and enter the tracking URL in the address bar to track your Spark on MaxCompute job.

□

2. Find the driver you want to view, and then click `stdout` in the Logs column.

□

7.7. Access instances in a VPC from Spark on MaxCompute

This topic describes how to access instances in a VPC from Spark on MaxCompute.

Spark VPC

VPC access

You can access instances (for example ECS, ApsaraDB for HBase, and ApsaraDB for RDS) in a VPC, or access user-defined private domain names from Spark on MaxCompute.

When you access instances in a VPC from Spark on MaxCompute, add the

`spark.hadoop.odps.cupid.vpc.domain.list` parameter to the `spark-defaults.conf` file or the `DataWorks` file to specify one or more instances. The value of this parameter is in the JSON format. When you configure this parameter, you must delete the spaces and line breaks between multiple lines in the parameter and merge JSON text into one line.

When you access different instances from Spark on MaxCompute, set

`spark.hadoop.odps.cupid.vpc.domain.list` to a required value, as described in the following examples. You must replace the RegionID, VPCID, instance domain name, and port number with actual values. For information about RegionID of a region, see .

Notice

- You must add the CIDR block 100.104.0.0/16 to the whitelist of the service that you want to access.
- If RegionID is `cn-shanghai` or `cn-beijing`, you must set `spark.hadoop.odps.cupid.smartnat.enable` to `true`.
- You can access only the services in one VPC in the current region from Spark on MaxCompute.

Example 1: access ApsaraDB for MongoDB

When you access ApsaraDB for MongoDB from Spark on MaxCompute, set

`spark.hadoop.odps.cupid.vpc.domain.list` , as shown in the following example. This ApsaraDB for MongoDB instance has a primary instance and a secondary instance.

```
{
  "regionId": "cn-beijing",
  "vpcs": [
    {
      "vpcId": "vpc-2zeaeq21mb1dmkqh0****",
      "zones": [
        {
          "urls": [
            {
              "domain": "dds-2ze3230cfea08****.mongodb.rds.aliyuncs.com",
              "port": 3717
            },
            {
              "domain": "dds-2ze3230cfea08****.mongodb.rds.aliyuncs.com",
              "port": 3717
            }
          ]
        }
      ]
    }
  ]
}
```

Results of merging JSON text into one line:

```
{"regionId": "cn-beijing", "vpcs": [{"vpcId": "vpc-2zeaeq21mb1dmkqh0****", "zones": [{"urls": [{"domain": "dds-2ze3230cfea08****.mongodb.rds.aliyuncs.com", "port": 3717}, {"domain": "dds-2ze3230cfea08****.mongodb.rds.aliyuncs.com", "port": 3717}]}]}
```

Example 2: access ApsaraDB for RDS

When you access ApsaraDB for RDS from Spark on MaxCompute, set `spark.hadoop.odps.cupid.vpc.domain.list`, as shown in the following example:


```
{
  "regionId": "cn-beijing",
  "vpcs": [
    {
      "vpcId": "vpc-2zeaeq21mb1dmkqh0****",
      "zones": [
        {
          "urls": [
            {
              "domain": "rm-2zem49k73c54z****.mysql.rds.aliyuncs.com",
              "port": 3306
            }
          ]
        }
      ]
    }
  ]
}
```

Results of merging JSON text into one line:

```
{"regionId": "cn-beijing", "vpcs": [{"vpcId": "vpc-2zeaeq21mb1dmkqh0****", "zones": [{"urls": [{"domain": "rm-2zem49k73c54z****.mysql.rds.aliyuncs.com", "port": 3306}]}]}
```

Example 3: access ApsaraDB for HBase

When you access ApsaraDB for HBase from Spark on MaxCompute, set `spark.hadoop.odps.cupid.vpc.domain.list`, as shown in the following example:

```
{
  "regionId": "cn-beijing",
  "vpcs": [
    {
      "vpcId": "vpc-2zeaeq21mb1dmkqh0exox",
      "zones": [
        {
          "urls": [
            {
              "domain": "hb-2zecxg2ltnpeg8me4-master*-***.hbase.rds.aliyuncs.com",
              "port": 2181
            }
          ],
          {

```

```
"domain":"hb-2zecxg2ltnpeg8me4-master*-.***.hbase.rds.aliyuncs.com",
"port":16000
},
{
"domain":"hb-2zecxg2ltnpeg8me4-master*-.***.hbase.rds.aliyuncs.com",
"port":16020
},
{
"domain":"hb-2zecxg2ltnpeg8me4-master*-.***.hbase.rds.aliyuncs.com",
"port":2181
},
{
"domain":"hb-2zecxg2ltnpeg8me4-master*-.***.hbase.rds.aliyuncs.com",
"port":16000
},
{
"domain":"hb-2zecxg2ltnpeg8me4-master*-.***.hbase.rds.aliyuncs.com",
"port":16020
},
{
"domain":"hb-2zecxg2ltnpeg8me4-master*-.***.hbase.rds.aliyuncs.com",
"port":2181
},
{
"domain":"hb-2zecxg2ltnpeg8me4-master*-.***.hbase.rds.aliyuncs.com",
"port":16000
},
{
"domain":"hb-2zecxg2ltnpeg8me4-master*-.***.hbase.rds.aliyuncs.com",
"port":16020
},
{
"domain":"hb-2zecxg2ltnpeg8me4-cor*-.***.hbase.rds.aliyuncs.com",
"port":16020
},
{
"domain":"hb-2zecxg2ltnpeg8me4-cor*-.***.hbase.rds.aliyuncs.com",
"port":16020
},
{
"domain":"hb-2zecxg2ltnpeg8me4-cor*-.***.hbase.rds.aliyuncs.com",
```

```

        "port":16020
    }
]
}
]
}
]
}
}

```

Results of merging JSON text into one line:

```

{"regionId":"cn-beijing","vpcs":[{"vpId":"vpc-2zeaeq21mb1dmkqh0exox","zones":[{"urls":[{"domain":"hb-2zecxg2ltmpeg8me4-master*-.***.hbase.rds.aliyuncs.com","port":2181},{domain":"hb-2zecxg2ltmpeg8me4-master*-.***.hbase.rds.aliyuncs.com","port":16000},{domain":"hb-2zecxg2ltmpeg8me4-master*-.***.hbase.rds.aliyuncs.com","port":16020},{domain":"hb-2zecxg2ltmpeg8me4-master*-.***.hbase.rds.aliyuncs.com","port":2181},{domain":"hb-2zecxg2ltmpeg8me4-master*-.***.hbase.rds.aliyuncs.com","port":16000},{domain":"hb-2zecxg2ltmpeg8me4-master*-.***.hbase.rds.aliyuncs.com","port":16020},{domain":"hb-2zecxg2ltmpeg8me4-master*-.***.hbase.rds.aliyuncs.com","port":2181},{domain":"hb-2zecxg2ltmpeg8me4-master*-.***.hbase.rds.aliyuncs.com","port":16000},{domain":"hb-2zecxg2ltmpeg8me4-master*-.***.hbase.rds.aliyuncs.com","port":16020},{domain":"hb-2zecxg2ltmpeg8me4-cor*-.***.hbase.rds.aliyuncs.com","port":16020},{domain":"hb-2zecxg2ltmpeg8me4-cor*-.***.hbase.rds.aliyuncs.com","port":16020},{domain":"hb-2zecxg2ltmpeg8me4-cor*-.***.hbase.rds.aliyuncs.com","port":16020}]}]}]}

```

Example 4: access ApsaraDB for Redis

When you access ApsaraDB for Redis from Spark on MaxCompute, set

`spark.hadoop.odps.cupid.vpc.domain.list` , as shown in the following example:

```
{
  "regionId": "cn-beijing",
  "vpcs": [
    {
      "vpcId": "vpc-2zeaeq21mb1dmkqh0****",
      "zones": [
        {
          "urls": [
            {
              "domain": "r-2zebda0d3c05****.redis.rds.aliyuncs.com",
              "port": 3717
            }
          ]
        }
      ]
    }
  ]
}
```

Results of merging JSON text into one line:

```
{"regionId": "cn-beijing", "vpcs": [{"vpcId": "vpc-2zeaeq21mb1dmkqh0****", "zones": [{"urls": [{"domain": "r-2zebda0d3c05****.redis.rds.aliyuncs.com", "port": 3717}]}]}}
```

Example 5: access LogHub

When you access LogHub from Spark on MaxCompute, set `spark.hadoop.odps.cupid.vpc.domain.list`, as shown in the following example:

```
{
  "regionId": "cn-beijing",
  "vpcs": [
    {
      "zones": [
        {
          "urls": [
            {
              "domain": "cn-beijing-intranet.log.aliyuncs.com",
              "port": 80
            }
          ]
        }
      ]
    }
  ]
}
```

Results of merging JSON text into one line:

```
{"regionId": "cn-beijing", "vpcs": [{"zones": [{"urls": [{"domain": "cn-beijing-intranet.log.aliyuncs.com", "port": 80}]}]}}
```

For the domain parameter, use the LogHub endpoint of the classic network or the VPC endpoint. For the endpoint of each region, see [Endpoints](#).

Example 6: access DataHub

When you access DataHub from Spark on MaxCompute, set

`spark.hadoop.odps.cupid.vpc.domain.list` , as shown in the following example:

```
{
  "regionId": "cn-beijing",
  "vpcs": [
    {
      "zones": [
        {
          "urls": [
            {
              "domain": "dh-cn-beijing.aliyun-inc.com",
              "port": 80
            }
          ]
        }
      ]
    }
  ]
}
```

Results of merging JSON text into one line:

```
{"regionId": "cn-beijing", "vpcs": [{"zones": [{"urls": [{"domain": "dh-cn-beijing.aliyun-inc.com", "port": 80}]}]}
```

For the domain parameter, use the ECS endpoint of the classic network as the DataHub endpoint.

Example 7: access a user-defined domain name

Assume that you customize a domain name `a.b.com` in a VPC. An access is initiated from Spark on MaxCompute by using this domain name and port `a.b.com:80`. You must complete the following configurations before the access:

1. Associate a zone with a VPC in PrivateZone.
2. Click **Authorize** to grant MaxCompute the read-only access permissions on PrivateZone.
3. In Spark node configuration, add the following two parameters:

```
spark.hadoop.odps.cupid.pvtz.rolearn=acs:ram:xxxxxxxxxx:role/aliyunodpsdefaultrole
spark.hadoop.odps.cupid.vpc.usepvtz=true
```

The `spark.hadoop.odps.cupid.pvtz.rolearn` parameter indicates your ARN information. You can obtain the information from the **RAM console**.

4. In the Spark configuration file, set `spark.hadoop.odps.cupid.vpc.domain.list`, as shown in the following example:

```
{
  "regionId": "cn-beijing",
  "vpcs": [
    {
      "vpcId": "vpc-2zeaeq21mb1dmkqh0****",
      "zones": [
        {
          "urls": [
            {
              "domain": "a.b.com",
              "port": 80
            }
          ],
          "zoneId": "9b7ce89c6a6090e114e0f7c415ed****"
        }
      ]
    }
  ]
}
```

Results of merging JSON text into one line:

```
{"regionId": "cn-beijing", "vpcs": [{"vpcId": "vpc-2zeaeq21mb1dmkqh0****", "zones": [{"urls": [{"domain": "a.b.com", "port": 80}], "zoneId": "9b7ce89c6a6090e114e0f7c415ed****"}]}}
```

Example 8: access HDFS

- To use HDFS, add *hdfs-site.xml*. The file contains the following content:

```
<? xml version="1.0"? >
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>dfs://DfsMountpointDomainName:10290</value>
  </property>
  <property>
    <name>fs.dfs.impl</name>
    <value>com.alibaba.dfs.DistributedFileSystem</value>
  </property>
  <property>
    <name>fs.AbstractFileSystem.dfs.impl</name>
    <value>com.alibaba.dfs.DFS</value>
  </property>
</configuration>
```

- In the Spark configuration file, set `spark.hadoop.odps.cupid.vpc.domain.list`, as shown in the following example:

```
{
  "regionId": "cn-shanghai",
  "vpcs": [{
    "vpcId": "vpc-xxxxxx",
    "zones": [{
      "urls": [{
        "domain": "DfsMountpointDomainName",
        "port": 10290
      }]
    }]
  }]
}
```

Results of merging JSON text into one line:

```
{"regionId": "cn-shanghai", "vpcs": [{"vpcId": "vpc-xxxxxx", "zones": [{"urls": [{"domain": "DfsMountpoi
ntDomainName", "port": 10290}]}]}}
```

7.8. Configure Spark on MaxCompute to access OSS resources

This topic describes how to configure Spark on MaxCompute to access Object Storage Service (OSS) resources.

Configure the OSS endpoint

Use the public endpoint of OSS in the target region when you debug features. Use the OSS endpoint in the virtual private cloud (VPC) if MaxCompute is deployed in a production environment. For more information, see [Regions and endpoints](#).

Configure the OSS access method

- Access OSS by using the AccessKey ID and AccessKey secret of an account.

```
spark.hadoop.fs.oss.accessKeyId = xxxxxx
spark.hadoop.fs.oss.accessKeySecret = xxxxxx
spark.hadoop.fs.oss.endpoint = oss-xxxxxx-internal.aliyuncs.com
```

- Access OSS by using a Security Token Service (STS) token.

Spark on MaxCompute can access OSS by using the AccessKey ID and AccessKey secret of an account. In this case, you must write the plaintext AccessKey ID and AccessKey secret in the configuration, which incurs security risks. We recommend that you configure Spark on MaxCompute to access OSS by using an STS token.

- Go to the [Cloud Resource Access Authorization](#) page and click Confirm Authorization Policy. Then, the MaxCompute project can access OSS resources of the current Alibaba Cloud account by using an STS token.

 **Note** You can authorize a MaxCompute project to access OSS resources by using this method only when the owner of the MaxCompute project is an Alibaba Cloud account that owns the OSS resources to be accessed.

- Obtain the Alibaba Cloud Resource Name (ARN) of the role that Spark on MaxCompute assumes.
 - Log on to the [RAM console](#).
 - In the left-side navigation pane, click **RAM Roles**.
 - On the **RAM Roles** page, search for **AliyunODPSDefaultRole**.
 - In the search result, click **AliyunODPSDefaultRole** in the **RAM Role Name** column. On the page that appears, obtain the value of **ARN** in the **Basic Information** section. The value is in the `acs:ram::xxxxxxxxxxxxxxxx:role/aliyunodpsdefaultrole` format.
- Add the following content to the configurations of Spark on MaxCompute:

```
# Configure Spark on MaxCompute to access OSS resources by using an STS token.
spark.hadoop.fs.oss.credentials.provider=org.apache.hadoop.fs.aliyun.oss.AliyunStsTokenCred
entialsProvider

# Configure the ARN of the role that Spark on MaxCompute assumes.
spark.hadoop.fs.oss.sts.token.roleArn=acs:ram::xxxxxxxxxxxxxxxx:role/aliyunodpsdefaultrole

# Configure the OSS endpoint in the VPC.
spark.hadoop.fs.oss.endpoint=oss-cn-hangzhou-internal.aliyuncs.com
```

Configure a whitelist

```
spark.hadoop.odps.cupid.vpc.domain.list = {"regionId":"xxxxxx", "vpcs":[{"zones":[{"urls":[{"domain":"bucketname.oss-xxxxxx-internal.aliyuncs.com", "port":80} ] } ]}]}
```

8. Interactive SQL (Lightning)

8.1. Overview

MaxCompute Lightning provides interactive query services for MaxCompute, and supports easy connection to MaxCompute projects based on the PostgreSQL protocol and syntax. This service allows you to quickly query and analyze MaxCompute project data using standard SQL and commonly used tools.

You can use major BI tools, such as Tableau and FineReport, to easily connect to MaxCompute projects, and perform BI analysis or ad hoc queries. The quick query feature in MaxCompute Lightning allows you to provide services by encapsulating project table data in APIs, supporting diverse application scenarios without data migration.

MaxCompute Lightning offers serverless computing services. No infrastructure is required and you pay only for queries.

Key features

- Compatibility with the PostgreSQL protocol

MaxCompute Lightning provides Java Database Connectivity (JDBC) or Open Database Connectivity (ODBC) interfaces that are compatible with the PostgreSQL protocol. Tools or applications based on PostgreSQL databases can easily be connected to MaxCompute projects using default drivers. The easy connection enables diverse PostgreSQL tools to be used for analyzing MaxCompute project data.

- Improved performance

Quick query for MaxCompute tables is optimized, especially for small datasets and high query concurrency, supporting diverse application scenarios, such as regular reports and service APIs.

- Unified permissions management

MaxCompute Lightning is a product designed for MaxCompute products and provides access to MaxCompute projects. This service shares the same access control system with MaxCompute projects. This ensures that users can only query data that they are authorized to access.

- Out-of-the-box feature and pay by queries

MaxCompute Lightning provides serverless computing services based on existing MaxCompute computing resources. To perform queries, you only need to establish connections to MaxCompute projects using MaxCompute Lightning.

You do not need to configure, manage, or maintain MaxCompute Lightning resources. When using MaxCompute Lightning, you only incur costs for the data amount processed for each query.

System architecture

MaxCompute Lightning provides a method of connecting endpoints, clients, or applications to JDBC or ODBC interfaces using PostgreSQL drivers. This enables secure data access within the unified access control system for MaxCompute projects.

Query tasks, connected and submitted by JDBC or ODBC interfaces, use serverless computing resources of MaxCompute Lightning to ensure query service quality.

□

Scenarios

- Ad hoc query

The query for small datasets (less than 100 GB) is optimized to allow you to easily query MaxCompute tables with low latency. You do not need to import the MaxCompute data into the AnalyticDB (ADS), Relational Database Service (RDS), or other systems, which reduces required resources and administration costs.

This scenario has the following characteristics: flexible data objects for queries, complicated logic, quick query, easy adjustment of query logic, and low latency query requirements within one minute. Users are often data analysts who master SQL skills and want to use familiar client tools for query analysis.

- Reporting and analysis

Analysis reports are generated based on the MaxCompute project data consolidated in the Extract-Transform-Load (ETL) process. The reports are provided to managers and business users for regular checks.

This scenario has the following characteristics: The queried data objects are usually the aggregated data. The queried data objects are included in small datasets. Queries are based on fixed and simple query logic. The scenario has low latency requirements. Latency for most queries is within 5 seconds. The query latency period varies greatly depending on the data volume and query complexity.

- Online application

MaxCompute project data can be encapsulated in RESTful APIs to support online applications.

In this scenario, MaxCompute Lightning serves as an accelerated query engine to provide MaxCompute table data as API services with the least amount of manual intervention. This is enabled by integrating **data service components** of Alibaba Cloud DataWorks.

8.2. Constraints and limitations

This article introduces you to the constraints and limitations of using the MaxCompute Lightning service.

DDL/DML constraints and limitations

MaxCompute Lightning only supports Select queries for MaxCompute tables and does not support UPDATE, CREATE, DELETE, and INSERT operations on MaxCompute tables.

Query constraints and limitations

- A maximum number of 1,024 scanned partitions can be queried when you query partitioned tables.
- Currently, views cannot be created or used.
- Currently, MAP、ARRAY、TINYINT、BINARY、TIMESTAMP and DECIMAL with accuracy data types are not supported.
- A maximum of 1 TB data can be scanned for a table in each query.
- The size of the submitted query statement cannot exceed 100 KB.
- The query timeout period is one hour.

UDF constraints and limitations

- User-defined functions (UDF) created using MaxCompute cannot be used in MaxCompute Lightning.
- **PostgreSQL** user-defined functions cannot be created or used in MaxCompute Lightning.
- MaxCompute built-in functions are not supported at this time.

Query concurrency constraints

A maximum of 20 concurrent queries for a MaxCompute project is supported by MaxCompute Lightning.

8.3. Quick Start

This topic describes how to access MaxCompute Lightning services with major third-party tools, including how to view tables of a specified MaxCompute project, and how to perform BI analysis.

Prerequisites

- Activate MaxCompute and create a project.

Using MaxCompute Lightning requires that MaxCompute has been activated and a project has been created.

If you have not activated MaxCompute, activate the service first. For more information, see **Activate MaxCompute**. Then, create a MaxCompute project.

- Create a table and import data.

Tables have been created in the project and data has been loaded. For more information, see **MaxCompute Quick Start**.

- Obtain account information.

The access ID and access key for the MaxCompute project have been obtained.

You can log on to the Alibaba Cloud website, and click **Console** to view the **AccessKey** page. Contact the owner of the primary account if your RAM user is not granted permission to view **AccessKey**. You also need to ensure your RAM user is granted permission to view project tables.

Prepare client tools for connection

MaxCompute Lightning is compatible with PostgreSQL interfaces and is accessible to client tools that are connected to PostgreSQL databases.

Tableau Desktop BI tools are used in this tutorial. Download related tools from the Tableau official website.

Other commonly used client tools, such as SQL Workbench/J, PSQL, FineReport BI, and MicroStrategy BI tools, can be connected to MaxCompute Lightning in the same way as to PostgreSQL databases.

Access services and perform analysis

After the connection service is successful, you can view the data table under the specified MaxCompute project for BI analysis.

1. Select PostgreSQL when establishing a connection to a server.

Start Tableau Desktop. In the left-side navigation pane, select **Select > To Servers > More > PostgreSQL**.

2. Enter service connection and user authentication information.

Parameter	Description
Server	Enter the MaxCompute Lightning endpoint of a specified region in the Server field. For example, enter the value <code>lightning.cn-shanghai.maxcompute.aliyun.com</code> as the endpoint for the China East 2 region.
	443
Database	MaxCompute project name
ID Verification	User name and password
Username/Password	User Access Key ID/Access Key Secret
SSL connection	Select the SSL connection check box.

3. Obtain project table information and create a data source or model.

After you configure the contact information and log on to the Tableau Desktop, this software loads tables of the connected MaxCompute project. You can choose tables to create data models and charts as required.

The following figure shows an example of a chart created based on required dimensions and measures.

Now you have gained access to MaxCompute Lightning using Tableau Desktop. You can perform BI analysis on the data of the connected MaxCompute projects.

 **Note** For better performance, it is recommended that you customize the connection to the Lightning data source using the TDC file supported by Tableau. For more information, see [Tableau Desktop](#).

8.4. Access domain name

MaxCompute Lightning provides region-specific endpoints that allow you to access MaxCompute Lightning services in the corresponding regions.

The following tables describe the MaxCompute Lightning service connection status in different regions and public cloud network environments.

Service connection status in different regions with external network

Region	Service status	External network endpoint
China East 1	In service	<code>lightning.cn-hangzhou.maxcompute.aliyun.com</code>
China East 2	In service	<code>lightning.cn-shanghai.maxcompute.aliyun.com</code>

Region	Service status	External network endpoint
China North 2	In service	lightning.cn-beijing.maxcompute.aliyun.com
Southern China 1	In service	lightning.cn-shenzhen.maxcompute.aliyun.com
China (Hong Kong)	Beta	lightning.cn-hongkong.maxcompute.aliyun.com
Asia Pacific SE 1	Beta	lightning.ap-southeast-1.maxcompute.aliyun.com
Other regions	Not activated	-

Service connection status in different regions with classic network

Region	Service status	Classic network endpoint
China East 1	In service	lightning.cn-hangzhou.maxcompute.aliyun-inc.com
China East 2	In service	lightning.cn-shanghai.maxcompute.aliyun-inc.com
China North 2	In service	lightning.cn-beijing.maxcompute.aliyun-inc.com
Southern China 1	In service	lightning.cn-shenzhen.maxcompute.aliyun-inc.com
China (Hong Kong)	Beta	lightning.cn-hongkong.maxcompute.aliyun-inc.com
Asia Pacific SE 1	Beta	lightning.ap-southeast-1.maxcompute.aliyun-inc.com
Other regions	Not activated	-

Service connection status in different regions with VPC network

Region	Service status	VPC endpoint
China East 1	In service	lightning.cn-hangzhou.maxcompute.aliyun-inc.com
China East 2	In service	lightning.cn-shanghai.maxcompute.aliyun-inc.com
China East 2 Financial Cloud	In service	lightning.cn-shanghai-finance.maxcompute.aliyun-inc.com
China North 2	In service	lightning.cn-beijing.maxcompute.aliyun-inc.com
Southern China 1	In service	lightning.cn-shenzhen.maxcompute.aliyun-inc.com
China (Hong Kong)	Beta	lightning.cn-hongkong.maxcompute.aliyun-inc.com
Asia Pacific SE 1	Beta	lightning.ap-southeast-1.maxcompute.aliyun-inc.com
Other regions	Not activated	-

8.5. Access services using JDBC interfaces

The MaxCompute Lightning query engine is based on PostgreSQL 8.2 and currently only supports SELECT queries for existing MaxCompute tables. For more information about the [query syntax](#) and [functions](#).

If no data has been added to MaxCompute projects or existing data needs to be processed, see the [MaxCompute help document](#). You can use the [MaxCompute client](#) or [DataWorks](#) to access MaxCompute projects for creating and processing data objects.

8.5.1. JDBC driver

MaxCompute provides JDBC interfaces that are fully compatible with the PostgreSQL protocol. Users can connect SQL client tools to the MaxCompute Lightning service using JDBC interfaces.

MaxCompute Lightning can be accessed using JDBC drivers from the PostgreSQL official website or other drivers optimized for MaxCompute Lightning.

1. [JDBC](#) drivers from the PostgreSQL official website.

 **Note** Many client tools already have PostgreSQL database drivers built in, you can use the built-in drivers. If it is not integrated, you can download required drivers from the PostgreSQL official website. Take the SQL Workbench/J client as an example. You can choose the PostgreSQL official drivers when creating a connection.

2. [JDBC drivers](#) optimized by Alibaba Cloud MaxCompute Lightning

The downloaded MaxCompute Lightning JDBC driver is saved as a MaxComputeLightningJDBC.jar file. Take the SQL Workbench/J client as an example. In the **Driver Management** menu, add the MaxCompute Lightning JDBC driver entry.

When you create a connection, select the MaxCompute Lightning JDBC driver that you just added from the Driver list.

8.5.2. Configure JDBC connections

To connect SQL client tools to MaxCompute projects, you must have JDBC URLs for the MaxCompute projects.

The following shows the format of a JDBC URL:

```
jdbc:postgresql://endpoint:port/database
```

The following table describes the connection parameters:

Parameter	Value	Description

Parameter	Value	Description
endpoint	Access domain name of MaxCompute Lightning in the region	For more information, see Access domain name . For example, accessing the Shanghai Region service through the external network using <code>lightning.cn-shanghai.maxcompute.aliyun.com</code>
port	443	-
database	Name of a MaxCompute project	-
User	Access Key ID of the user	-
password	Access Key Secret of the user	-
ssl	true	MaxCompute Lightning servers are enabled with SSL protection by default, and you must use SSL connections.
prepareThreshold	0	Optional. When using the JDBC PrepareStatement function, it is recommended to set <code>prepareThreshold=0</code> .

For example, `jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/myproject`

You must specify the user, password, and SSL connection parameters before establishing a connection to MaxCompute projects.

You can also add parameters to the JDBC URL to connect to MaxCompute projects. For example:

```
jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/myproject? ssl=true& prepareThreshold=0&user=xxx&password=yyy
```

Note:

- `lightning.cn-shanghai.maxcompute.aliyun.com`: The endpoint of the China East 2 region.
- `Myproject`: The name of the MaxCompute project you want to access.
- `SSL=true`: The application of SSL connections.
- `xxx`: Access Key ID of the user.
- `yyy`: Access Key Secret of the user.

8.5.3. Access services using common tools

The following sections use major client tools, such as SQL Workbench/J, PSQL, and Tableau BI tools, as examples to describe how to access MaxCompute Lightning. Other commonly used tools can be connected to MaxCompute Lightning in the same way as to PostgreSQL databases.

Alibaba Cloud Quick BI

1. Log on Quick BI console, click **Data source** in the left-side navigation pane.

2. On the data source management page, click the **Create data source** in the upper-right corner.
3. Select PostgreSQL in the cloud database or external data source, and add a data source.
4. In the dialog box that appears, enter the connection information for MaxCompute Lightning. Then, test the connection.

Parameter	Description
Database address	Enter the endpoint for the region of MaxCompute Lightning. You can enter the endpoint for a public network, classic network, or VPC network.
Database	Enter the name of the to-be-accessed MaxCompute project.
Schema	MaxCompute project name
User name/Password	User Access Key ID/Access Key Secret.

SQL Workbench/J

SQL Workbench/J is a widely used free and cross-platform SQL query tool. This tool can be connected to MaxCompute Lightning using the PostgreSQL driver.

1. [Download](#) and install SQL Workbench/J.
2. Start SQL Workbench/J, establish a database connection.

Select the PostgreSQL driver, connect SQL Workbench/J to the MaxCompute Lightning URL of a project. You must enter the Access Key ID and Access Key Secret of the user.

 - Alternatively, you can click **Extended Properties** and set `ssl` to `true` in the displayed dialog box.
3. After SQL Workbench/J is connected to MaxCompute Lightning, you can view, query, and analyze the table data in the SQL Workbench/J workspace.

psql

The `psql` is a PostgreSQL interactive terminal that enables you to perform queries using commands. The clients of `psql` are installed by default when PostgreSQL databases are installed in a local PC.

You can connect `psql` to MaxCompute Lightning using `psql` commands. The syntax for the connection is the same as that for the connection to the PostgreSQL database.

```
psql -h <endpoint> -U <userid> -d <databasename> -p <port>
```

Parameter description:

- `<endpoint>`: The endpoint of MaxCompute Lightning. For more information, see [Access domain name](#).
- `<userid>`: Access Key ID.
- `<databasename>`: MaxCompute project name.

- <port>: 443

After the command is executed, enter the <userid> password (Access Key Secret) in the command prompt.

Example:

□

 **Note** SSL connections are preferred for psql by default.

Tableau Desktop

Start BI tools, select the PostgreSQL data source, and configure the connection.

When you configure the connection, select the **SSL Connection** check box.

After logging on to Tableau Desktop, you can create charts for visual analysis.

 **Note** For better performance, it is recommended that you customize the connection to the Lightning data source using the TDC file supported by Tableau. Procedure:

1. Save the following xml content as a postgresql.tdc file.

```
<?xml version='1.0' encoding='utf-8' ?>
<connection-customization class='postgres' enabled='true' version='8.10'>
  <vendor name='postgres' />
  <driver name='postgres' />
  <customizations>
    <customization name='CAP_CREATE_TEMP_TABLES' value='no' />
    <customization name='CAP_STORED_PROCEDURE_TEMP_TABLE_FROM_BUFFER' value='no' />
    <customization name='CAP_CONNECT_STORED_PROCEDURE' value='no' />
    <customization name='CAP_SELECT_INTTO' value='no' />
    <customization name='CAP_SELECT_TOP_INTTO' value='no' />
    <customization name='CAP_ISOLATION_LEVEL_SERIALIZABLE' value='yes' />
    <customization name='CAP_SUPPRESS_DISCOVERY_QUERIES' value='yes' />
    <customization name='CAP_SKIP_CONNECT_VALIDATION' value='yes' />
    <customization name='CAP_ODBC_TRANSACTIONS_SUPPRESS_EXPLICIT_COMMIT' value='yes' />
    <customization name='CAP_ODBC_TRANSACTIONS_SUPPRESS_AUTO_COMMIT' value='yes' />
    <customization name='CAP_ODBC_REBIND_SKIP_UNBIND' value='yes' />
    <customization name='CAP_FAST_METADATA' value='no' />
    <customization name='CAP_ODBC_METADATA_SUPPRESS_SELECT_STAR' value='yes' />
    <customization name='CAP_ODBC_METADATA_SUPPRESS_EXECUTED_QUERY' value='yes' />
    <customization name='CAP_ODBC_UNBIND_AUTO' value='yes' />
    <customization name='SQL_TXN_CAPABLE' value='0' />
    <customization name='CAP_ODBC_CURSOR_FORWARD_ONLY' value='yes' />
    <customization name='CAP_ODBC_TRANSACTIONS_COMMIT_INVALIDATES_PREPARED_QUERY' value='yes' />
  </customizations>
</connection-customization>
```

2. Save the file to the `\My Documents\My Tableau Repository\Datasources` directory. If it is Tableau Server, save it in `C:\ProgramData\Tableau\Tableau Server\data\tabsvc\vizqlserver\Datasources` under Windows, and save it in `/var/opt/tableau/tableau_server/data/tabsvc/vizqlserver/Datasources/` under Linux. .
3. Reopen Tableau and use the PostgreSQL data source to connect to the MaxCompute Lightning service. For more information about custom data sources for tdc files, see [official Tableau documentation](#).

FineReport

1. Start FineReport, and select **Server > Define database connection**.
2. Add a JDBC connection.

The configurations are described as follows:

Parameter	Description
Database	Postgre
Driver	org.postgresql.Driver that is integrated in FineReport
URL	<pre>jdbc:postgresql://<MaxCompute Lightning Endpoint>:443/<Project_Name>? ssl=true&prepareThreshold=0</pre> For example, <code>jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/lightning_demo? ssl=true&prepareThreshold=0</code>
User name/Password	User Access Key ID and Access Key Secret

8.6. Query syntax and built-in functions

Based on the official PostgreSQL functions, MaxCompute Lightning supports the query syntax and built-in functions, such as MAX_PT.

Query syntax

The MaxCompute Lightning query engine is based on PostgreSQL 8.2. MaxCompute Lightning only supports SELECT queries for existing MaxCompute tables. For more information about the query syntax, visit [PostgreSQL](#).

Function

MAX_PT

Syntax

```
max_pt(table_full_name)
```

Description

This function is used to return the name of and values in the level-1 partition that has the maximum partitioning column value and contains data files in a partitioned table. The returned values are sorted in alphabetic order.

Parameter

table_full_name: the table name, which must be of the STRING type. The table name must contain the project name, such as prj.src. You must have read permissions on this table.

Return value

The name of and values in the level-1 partition that has the maximum partitioning column value are returned.

Example

Assume that the tbl table is a partitioned table. The partitions in the table are as follows and contain data files:

```
pt ='20120901'  
pt ='20120902'
```

In this example, 20120902 is returned. The MaxCompute SQL statement reads the data in the pt='20120902' partition.

```
select * from tbl where pt=max_pt('myproject.tbl');
```

8.7. View tasks

MaxCompute Lightning provides a system view stv_recents. By querying the view, you can view all query tasks that the current user is running.

View running queries

You can execute the query command to view the related information, including query ID, user name, query SQL statement, start time, duration, and waiting resources. Note: The "t" indicates that a query task has not been executed yet and is waiting for resources. The "f" indicates that the resources are being acquired and that the query task is being executed.

Run the following query command.

```
select * from stv_recents;
```

The following figure shows a command output example.

□

Cancel running queries

You can obtain information on running queries by querying the stv_recents table. To cancel a running query, execute the following query command.

```
select cancel('query_id');
```

In parentheses is the query_id of a running query.

□

9.Graph

9.1. Summary

MaxCompute Graph is a processing framework designed for iterative graph computing. MaxCompute Graph jobs use graphs to build models. Graphs are composed of vertices and edges, which contain values.

MaxCompute Graph supports the following graph editing operations:

- Editing the value of Vertex or Edge.
- Add/delete Vertices.
- Add/delete Edges.

 **Note** When editing a vertex and an edge, you must maintain their relationship.

This process outputs a final solution after performing iterative graph editing and evolution. Typical applications include [PageRank](#), [SSSP algorithm](#), and [Kmeans algorithm](#). Use Java SDK, an interface provided by MaxCompute Graph to compile graph computing programs.

Graph Data structure

Graphs processed by MaxCompute Graph must be directed graphs consisting of vertices and edges. As MaxCompute only provides a two-dimensional storage structure, you must resolve graph data into two-dimensional tables and store them in MaxCompute.

During graph computing analysis, use custom GraphLoader to convert two-dimensional table data to vertices and edges in the MaxCompute Graph engine. You can determine how to resolve graph data into two-dimensional tables based on your service scenarios. In the sample code, the table formats correspond to different graph data structures.

The vertex structure can be described as < ID, Value, Halted, Edges >, which respectively indicates the vertex ID (ID), value (Value), status (Halted, indicating whether an iteration needs to be stopped), and edge set (Edges, indicating lists of all edges starting from the vertex). The edge structure is described as < DestVertexID, Value >, which respectively indicates the destination vertex (DestVertexID) and value (Value).

□

For example, the preceding figure consists of the following vertices:

Vertex	<ID, Value, Halted, Edges>
v0	<0, 0, false, [<1, 5 >, <2, 10 >] >
v1	<1, 5, false, [<2, 3>, <3, 2>, <5, 9>]>
v2	<2, 8, false, [<1, 2>, <5, 1 >]>
v3	<3, Long.MAX_VALUE, false, [<0, 7>, <5, 6>]>
v5	<5, Long.MAX_VALUE, false, [<3, 4 >]>

Graph program logic

Graph loading

The framework calls custom GraphLoader and resolves records of an input table to vertices or edges.

Distributed architecture: The framework calls custom Partitioner to partition vertices and distributes them to corresponding Workers. (Default partitioning logic: Calculates the hash value of a vertex ID and performs the modulo operation on the number of Workers.)

□

For example, assume in the preceding figure that the number of Workers is 2. v0 and v2 are allocated to Worker 0 because the result of the ID mod 2 is 0. v1, v3, and v5 are allocated to Worker 1 as the result of the ID mod 2 is 1.

Iteration calculation

- An iteration is called a **superstep**. It traverses all vertices in a non-halted status (the value of the halted is false) or all vertices that receive messages (a vertex in halted status is automatically activated after receiving a message), and calls their compute (ComputeContext context, Iterable messages) method.
- Follow these steps on your implemented compute (ComputeContext context, Iterable messages) method:
 - Process messages sent from the previous SuperStep to the current Vertex.
 - Edit graph as needed:
 - Revise value of Vertex/Edge
 - Send messages to certain Vertices
 - Add/Delete Vertex or Edge
 - Use Aggregator to collect information to update the global information.
 - Set the current vertex to a halted or non-halted status.
 - During iteration, the framework asynchronously sends messages to the corresponding Worker and processes the messages in the next SuperStep without your intervention.

Iteration termination

If any of the following conditions are met, iteration is terminated.

- All vertices are in the halted state (the value of Halted is true) and no new message is generated.
- A maximum number of iterations is reached.
- The terminate method of an Aggregator returns true.

The pseudocode is as follows:


```
// 1. load
for each record in input_table {
    GraphLoader.load();

// 2. setup
WorkerComputer.setup();
for each aggr in aggregators {
    aggr.createStartupValue();

for each v in vertices {
    v.setup();

// 3. superstep
for (step = 0; step < max; step ++ ) {
    for each aggr in aggregators {
        aggr.createInitialValue();

    for each v in vertices {
        v.compute();

// 4. cleanup
for each v in vertices {
    v.cleanup();

WorkerComputer.cleanup();
```

9.2. Aggregator

This article explains the implementation and related APIs of Aggregator and uses KmeansClustering as an example to illustrate the use of Aggregator.

In MaxCompute Graph, Aggregator helps to collect and process global information. In MaxCompute Graph, Aggregator is used to summarize and process global information.

Aggregator implementation

The logic of Aggregator is divided into the following two parts:

- One part is run on all Workers in distributed mode.
- The other part is only run on the Worker where AggregatorOwner is located in a single vertex mode.

Operations run on all Workers include creating an initial value and partial aggregation. The partial aggregation result is sent to the Worker where AggregatorOwner is located. The Worker then aggregates partial aggregation objects sent by common Workers to obtain a global aggregation result, and determines whether the iteration is ended or not. The global aggregation result is sent to all Workers over the next round of supersteps for the next iteration, as shown in the following figure.

□

Aggregator APIs

Aggregator provides five APIs for user implementation. The following section describes the call time and application of the five APIs.

- `createStartupValue(context)`

This API runs once on all Workers. It is called before all supersteps start, and is generally used to initialize `AggregatorValue`. In the first superstep iteration (superstep equals 0), the `AggregatorValue` object initialized by the API can be obtained by the call of `WorkerContext.getLastAggregatedValue()` or `ComputeContext.getLastAggregatedValue()`.

- `createInitialValue(context)`

This API is called once on all Workers when each superstep is initiated. It is used to initialize `AggregatorValue` for the current iteration. Generally, the result of the previous iteration is obtained through `WorkerContext.getLastAggregatedValue()`, and partial initialization is run.

- `aggregate(value, item)`

This API runs on all Workers. It is triggered by an explicit call of `ComputeContext#aggregate(item)`, while the preceding two APIs are automatically called by the framework. This API is used to run partial aggregation. The first parameter `value` indicates the result that the Worker has aggregated in the current superstep. (The initial value is the object returned by `createInitialValue`). The second parameter is transmitted when the user code calls `ComputeContext#aggregate(item)`. In this API, `item` is usually used to update `value` for aggregation. After all the aggregate operations are executed, the obtained value is the partial aggregation result of the Worker. Then, the result is sent by the framework to the Worker where `AggregatorOwner` is located.

- `merge(value, partial)`

This API runs by the Worker where `AggregatorOwner` is located. It is used to merge partial aggregation results of Workers to obtain the global aggregation object. Similar to `aggregate`, `value` indicates aggregated results, while `partial` indicates objects to be aggregated. `Partial` is used to update `value`.

For example, assume that three Workers `w0`, `w1`, and `w2` exist with the partial aggregation results of `p0`, `p1`, and `p2`. If `p1`, `p0`, and `p2` in sequence are sent to the Worker where the `AggregatorOwner` is located, then the merge sequence will be as follows:

- i. `merge(p1, p0)` runs first, and `p1` and `p0` are aggregated as `p1'`.
- ii. `merge(p1', p2)` runs, and `p1'` and `p2` are aggregated as `p1''`, which is the global aggregation result in this superstep.

The preceding example shows that execution of the `merge()` operation is not required when only one Worker exists. That is, `merge()` is not called.

- `terminate(context, value)`

After the Worker where AggregatorOwner is located runs `merge()`, the framework calls `terminate(context, value)` to perform the final processing. The second parameter value indicates the global aggregation result obtained by `merge()`. The global aggregation can be modified further in this method. After `terminate()` is run, the framework distributes global aggregation objects to all Workers for the next superstep. A special feature of `terminate()` is that if true is returned, iteration of the entire job ends. Otherwise, iteration continues. In machine learning scenarios, it is usually determined that a job ends when true is returned after convergence.

KmeansClustering example

The following section uses typical KmeansClustering as an example to describe how to use Aggregator. The following section uses KmeansClustering as an example to describe how to use the Aggregator.

 **Note** The complete code is provided in the Kmeans attachment. Here, the code is resolved in the following sequence.

- **GraphLoader section**

GraphLoader: The GraphLoader part is used to load an input table and convert it to a vertex or edge of a graph. Each row of data in the input table is a sample, a sample constructs a vertex, and `VertexValue` is used to store samples.

Initially, a writable class `KmeansValue` is defined as the `VertexValue` type:

```
public static class KmeansValue implements Writable {
    DenseVector sample;
    public KmeansValue() {

    }

    public KmeansValue(DenseVector v) {
        this.sample = v;
    }

    @Override
    public void write(DataOutput out) throws IOException {
        writeForDenseVector(out, sample);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        sample = readFieldsForDenseVector(in);
    }
}
```

KmeansValue: A `DenseVector` object is encapsulated in `KmeansValue` to store a sample. The `DenseVector` type is from `matrix-toolkits-java`. `writeForDenseVector()` and `readFieldsForDenseVector()` are used for serialization and deserialization. For more information, see the complete code in the Kmeans attachment.

The custom `KmeansReader` code is as follows:

```

public static class KmeansReader extends
    GraphLoader<LongWritable, KmeansValue, NullWritable, NullWritable> {

    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, KmeansValue, NullWritable, NullWritable> context)
        throws IOException {
        KmeansVertex v = new KmeansVertex();
        v.setId(recordNum);
        int n = record.size();
        DenseVector dv = new DenseVector(n);
        for (int i = 0; i < n; i++) {
            dv.set(i, ((DoubleWritable)record.get(i)).get());
        }

        v.setValue(new KmeansValue(dv));
        context.addVertexRequest(v);
    }
}

```

In KmeansReader, a vertex is created when each row of data (a record) is read. recordNum is used as the vertex ID, and the record content is converted to the DenseVector object and encapsulated in VertexValue.

- **Vertex**

Custom KmeansVertex code: Regarding its logic, partial aggregation is performed for samples maintained in each iteration. For more information about its logic, see implementation of Aggregator in the following section:

```

public static class KmeansVertex extends
    Vertex<LongWritable, KmeansValue, NullWritable, NullWritable> {

    @Override
    public void compute(
        ComputeContext<LongWritable, KmeansValue, NullWritable, NullWritable> context,
        Iterable<NullWritable> messages) throws IOException {
        context.aggregate(getValue());
    }
}

```

- **Aggregator**

The main logic of entire Kmeans is centralized in Aggregator. Custom KmeansAggrValue is used to maintain the content to be aggregated and distributed.

```

public static class KmeansAggrValue implements Writable {
    DenseMatrix centroids;
    DenseMatrix sums; // used to recalculate new centroids
    DenseVector counts; // used to recalculate new centroids
    @Override
    public void write(DataOutput out) throws IOException {
        writeForDenseDenseMatrix(out, centroids);
        writeForDenseDenseMatrix(out, sums);
        writeForDenseVector(out, counts);

        @Override
        public void readFields(DataInput in) throws IOException {
            centroids = readFieldsForDenseMatrix(in);
            sums = readFieldsForDenseMatrix(in);
            counts = readFieldsForDenseVector(in);
        }
    }
}

```

Three objects are maintained in KmeansAggrValue. centroids indicates the existing K centers. If the sample is m-dimensional, centroids is a matrix of K x m. sums is a matrix of the same size as centroids, and each element records the sum of a specific dimension of the sample closest to a specific center. For example, sums(i,j) indicates the sum of dimension j of the sample closest to center i.

counts is a K-dimensional vector, records the number of samples closest to each center. sums and counts are used together to calculate a new center, which is a main content of aggregation.

The next is KmeansAggregator used for custom Aggregator implementation. The following describes implementation in order of the preceding APIs.

- i. Run createStartupValue(), see the following:

```

public static class KmeansAggregator extends Aggregator<KmeansAggrValue> {
    public KmeansAggrValue createStartupValue(WorkerContext context) throws IOException {
        KmeansAggrValue av = new KmeansAggrValue();
        byte[] centers = context.readCacheFile("centers");
        String lines[] = new String(centers).split("\n");
        int rows = lines.length;
        int cols = lines[0].split(",").length; // assumption rows >= 1
        av.centroids = new DenseMatrix(rows, cols);
        av.sums = new DenseMatrix(rows, cols);
        av.sums.zero();
        av.counts = new DenseVector(rows);
        av.counts.zero();
        for (int i = 0; i < lines.length; i++) {
            String[] ss = lines[i].split(",");
            for (int j = 0; j < ss.length; j++) {
                av.centroids.set(i, j, Double.valueOf(ss[j]));
            }
        }

        return av;
    }
}

```

In the preceding method, a `KmeansAggrValue` object is initialized, the initial center is read from the resource file `centers`, and a value is granted to centroids. The initial values of sums and counts are 0.

ii. Run `createInitialValue()`, see the following:

```

@Override
public void aggregate(KmeansAggrValue value, Object item)
    throws IOException {
    DenseVector sample = ((KmeansValue)item).sample;
    // find the nearest centroid
    int min = findNearestCentroid(value.centroids, sample);
    // update sum and count
    for (int i = 0; i < sample.size(); i++) {
        value.sums.add(min, i, sample.get(i));
    }

    value.counts.add(min, 1.0d);
}

```

In the `createInitialValue()` method, `findNearestCentroid()` is called to find the index of the center that has the shortest Euclidean distance with the sample item. Then, each dimension is added to sums, and the value of counts is plus 1. For more information about how to implement `findNearestCentroid()`, see the `Kmeans` attachment.

The preceding three functions run on all Workers to implement partial aggregation. The following describes global aggregation-related operations that run on the Worker where AggregatorOwner is located.

1. Run merge:

```
@Override
public void merge(KmeansAggrValue value, KmeansAggrValue partial)
    throws IOException {
    value.sums.add(partial.sums);
    value.counts.add(partial.counts);
}
```

The implementation logic of merge is to add values of sums and counts aggregated by each Worker .

2. Run terminate():

```
@Override
public boolean terminate(WorkerContext context, KmeansAggrValue value)
    throws IOException {
    // Calculate the new means to be the centroids (original sums)
    DenseMatrix newCentroids = calculateNewCentroids(value.sums, value.counts, value.centroids);
    // print old centroids and new centroids for debugging
    System.out.println("\nsuperstep: " + context.getSuperstep() +
        "\nold centroid:\n" + value.centroids + " new centroid:\n" + newCentroids);
    boolean converged = isConverged(newCentroids, value.centroids, 0.05d);
    System.out.println("superstep: " + context.getSuperstep() + "/"
        + (context.getMaxIteration() - 1) + " converged: " + converged);
    if (converged || context.getSuperstep() == context.getMaxIteration() - 1) {
        // converged or reach max iteration, output centroids
        for (int i = 0; i < newCentroids.numRows(); i++) {
            Writable[] centroid = new Writable[newCentroids.numColumns()];
            for (int j = 0; j < newCentroids.numColumns(); j++) {
                centroid[j] = new DoubleWritable(newCentroids.get(i, j));
            }

            context.write(centroid);
        }

        // true means to terminate iteration
        return true;
    }

    // update centroids
    value.centroids.set(newCentroids);
    // false means to continue iteration
    return false;
}
```

In `terminate()`, `calculateNewCentroids()` is called based on sums and counts to calculate the average value and obtain the new center. Then, `isConverged()` is called based on the Euclidean distance between the new and old centers to determine whether the center has been converged. If the number of convergences or iterations reaches the upper threshold, the new center is output, and `true` is returned to end the iteration. Otherwise, the center is updated, and `false` is returned to continue iteration. For more information about how to implement `calculateNewCentroids()` and `isConverged()`, see the attachment.

- **main() method**

The `main()` method is used to build `GraphJob`, perform related settings, and submit a job. The code is as follows:

```
public static void main(String[] args) throws IOException {
    if (args.length < 2)
        printUsage();
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(KmeansReader.class);
    job.setRuntimePartitioning(false);
    job.setVertexClass(KmeansVertex.class);
    job.setAggregatorClass(KmeansAggregator.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    // default max iteration is 30
    job.setMaxIteration(30);
    if (args.length >= 3)
        job.setMaxIteration(Integer.parseInt(args[2]));
    long start = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
}
```

 **Note** If `job.setRuntimePartitioning(false)` is set to `false`, data loaded by each worker will not be partitioned based on `Partitioner`. That is, who loads the data maintains it.

Conclusion

This article introduces the aggregator features in the MaxCompute graph, the API meaning, and the kmeans Clustering example. To sum it up, Aggregator can be implemented as follows:

1. Each Worker runs `createStartupValue` during startup to create `AggregatorValue`.
2. Each Worker runs `createInitialValue` before each iteration initializes `AggregatorValue` in the current round.
3. In an iteration, each vertex uses `context.aggregate()` to run `aggregate()`, implementing partial iteration in the Worker.
4. Each Worker sends the partial iteration result to the Worker where `AggregatorOwner` is

located.

5. The Worker where AggregatorOwner is located runs merge several times to implement global aggregation.
6. The Worker where AggregatorOwner is located runs terminate to process the global aggregation result and determines whether to end the iteration.

Attachment

Kmeans

9.3. Limits

The limits of MaxCompute Graph are as follows:

- Each job can reference up to 256 resources. A table or an archive is considered as one unit (that is, one resource).
- The total number of bytes of resources referenced by one job cannot exceed 512 MB. Each job can reference up to 512 MB of bytes of resources.
- The number of inputs of one job cannot exceed 1,024. and the number of input tables cannot exceed 64. The number of outputs of one job cannot exceed 256.
- A label can be up to 256 characters in length and can contain letters, numbers, and special characters including underscores (_), pound signs (#), periods (.), and hyphens (-). Labels specified for multiple outputs cannot be null or empty strings.
- Each job can have up to 64 custom counters. The group name and counter name can be up to 100 characters in length. The names cannot contain pound signs (#).
- The number of Workers of one job is calculated by the framework. The maximum number is 1,000. If this threshold value is exceeded, an exception is thrown.
- One Worker occupies 200 resources of the CPU by default. The range is [50, 800].
- One Worker occupies 4096 MB of the memory by default. The range is [256 MB, 12 GB].
- A threshold for a Worker to read a resource repeatedly is 64.
- The split size can be set, however, as 64 MB is the by default size.. The range is $0 < \text{split\_size} \leq (9223372036854775807 >> 20)$.
- In the MaxCompute Graph program, GraphLoader/Vertex/Aggregator running in a cluster is restricted by the Java sandbox. (The main program of Graph jobs is not restricted.) For more information about the restrictions, see [Java sandbox](#).

9.4. Function overview

Running jobs

The MaxCompute console provides JAR commands to run MaxCompute Graph jobs. These commands are used the same way as [MapReduce JAR commands](#) run.

This article introduces you to these commands.

Usage: jar [<GENERIC_OPTIONS>] <MAIN_CLASS> [ARGS]

- conf <configuration_file> Specify an application configuration file
- classpath <local_file_list> classpaths used to run mainClass
- D <name>=<value> Property value pair, which is used to run mainClass
- local Run job in local mode
- resources <resource_name_list> file/table resources used in graph, separated by command

< GENERIC_OPTIONS> can be the following parameters (all are optional):

- -conf < configuration file >: Specifies the JobConf configuration file.
- -classpath < local_file_list >: Indicates the class path for local implementation. It is mainly used to specify the JAR package containing the main function.

The main function and Graph job are usually written in the same package, for example, in the Single Source Shortest Path (SSSP) package. Therefore, the -resources and -classpath parameters in the sample code both contain the JAR package. The difference is that -resources refers to the value of the Graph job and runs in a distributed environment, while -classpath refers to the main function and runs locally. The specified JAR package path is also a local file path. Package names are separated using system default file delimiters. Generally, the delimiter is a semicolon (;) in a Windows system and a comma (,) in a Linux system.

- -D < prop_name > = < prop_value >: Specifies the Java attributes of < mainClass > for local implementation. Multiple attributes can be defined.
- -local: Runs the Graph job in local mode, which is mainly used for program debugging.
- -resources <resource_name_list >: Indicates the resource statement used for Graph job running. Generally, the name of the resource where the Graph job is located must be specified in resource_name_list. If you read other MaxCompute resources in the Graph job, the resource names must be added to resource_name_list. Resource names are separated by commas (,). When resources are used across projects, PROJECT_NAME/resources/ must be prefixed. For example, -resources otherproject/resources/resfile; .

In addition, run the main function of the Graph job to directly submit a job to MaxCompute, rather than submitting a job through the MaxCompute console. The following section uses the [PageRank algorithm](#) as an example:

```

public static void main(String[] args) throws Exception {
    if (args.length < 2)
        printUsage();
    Account account = new AliyunAccount(accessId, accessKey);
    Odps odps = new Odps(account);
    odps.setEndpoint(endPoint);
    odps.setDefaultProject(project);
    SessionState ss = SessionState.get();
    ss.setOdps(odps);
    ss.setLocalRun(false);
    String resource = "mapreduce-examples.jar";
    GraphJob job = new GraphJob();
    // Add the JAR file in use and other files to class cache resource, corresponding to resources specified by -libjars in the JAR command
    job.addCacheResourcesToClassPath(resource);
    job.setGraphLoaderClass(PageRankVertexReader.class);
    job.setVertexClass(PageRankVertex.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    // default max iteration is 30
    job.setMaxIteration(30);
    if (args.length >= 3)
        job.setMaxIteration(Integer.parseInt(args[2]));
    long startTime = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - startTime) / 1000.0
        + " seconds");
}

```

Input and output

You cannot customize input and output formats.

The following example shows how to define a job input. Multiple inputs are supported:

```
GraphJob job = new GraphJob();
job.addInput(TableInfo.builder().tableName( "tblname" ).build()); //Table as input
job.addInput(TableInfo.builder().tableName( "tblname" ).partSpec("pt1=a/pt2=b").build()); //Shard a
s input
//Read-only columns col2 and col0 of the input table. In the load() method of GraphLoader, column col2
is obtained by record.get(0), and the sequence is the same
job.addInput(TableInfo.builder().tableName( "tblname" ).partSpec("pt1=a/pt2=b").build(), new String[
]{"col2", "col0"});
```

Note

- For more information about the job input definition, see the description of the `addInput()` method in a `GraphJob`. The framework reads records in the input table and transmits them to custom `GraphLoader` to load data.
- Limits: Currently, shard filtering conditions are not supported. For more information , see [Application restrictions](#).

The following example shows how to define a job output. Multiple job outputs are supported. Each output is marked by a label:

```
GraphJob job = new GraphJob();
//If the output table is a shard table, the last level of shards must be provided
job.addOutput(TableInfo.builder().tableName("table_name").partSpec("pt1=a/pt2=b").build());
// Parameter true indicates overwriting shards specified by tableinfo, that is, the meaning of INSERT O
VERWRITE. Parameter false indicates the meaning of INSERT INTO
job.addOutput(TableInfo.builder().tableName("table_name").partSpec("pt1=a/pt2=b").lable("output1")
.build(), true);
```

Note

- For more information about the job output definition, see the description of the `addOutput()` method in `GraphJob`.
- When a Graph job runs, records can be written to an output table using the `write()` method of `WorkerContext`. Labels must be specified for multiple outputs, such as "output1" in the preceding section.
- For more information, see [Application limits](#).

Read resources

- **Add resources to the graph program**

In addition to JAR commands, you can use the following two methods of `GraphJob` to specify resources read by Graph:

```
void addCacheResources(String resourceNames)
void addCacheResourcesToClassPath(String resourceNames)
```

- Use resources in the graph program

To read resources in the Graph program, follow these steps:

```
public byte[] readCacheFile(String resourceName) throws IOException;
        public Iterable<byte[]> readCacheArchive(String resourceName) throws IOExcep
tion;
        public Iterable<byte[]> readCacheArchive(String resourceName, String relativeP
ath)throws IOException;
        public Iterable<WritableRecord> readResourceTable(String resourceName);
public BufferedInputStream readCacheFileAsStream(String resourceName) throws IOException;
public Iterable<BufferedInputStream> readCacheArchiveAsStream(String resourceName) throws IOE
xception;
public Iterable<BufferedInputStream> readCacheArchiveAsStream(String resourceName, String rela
tivePath) throws IOException;
```

 Note

- Resources are generally read using the setup() method of WorkerComputer, stored in Worker Value, and obtained using the getWorkerValue() method.
- To reduce overall memory consumption, use the preceding stream APIs so that resources can be read and processed simultaneously.
- For more information , see [Application limits](#).

9.5. SDK summary

Maven users can search for odps-sdk-graph in the [Maven database](#) to get the required SDK (available in different versions). The configuration information is as follows:

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-graph</artifactId>
  <version>0.20.7</version>
</dependency>
```

Main interface	Description
GraphJob	GraphJob is inherited from JobConf and is used to define, submit, and manage a MaxCompute Graph job.

Main interface	Description
Vertex	A vertex is a node that is defined by the attributes including ID, value, halted, and edges. A vertex is implemented by the <code>setVertexClass</code> interface of <code>GraphJob</code> .
Edge	Edge is the abstract of edges in a graph, including the attributes <code>destVertexId</code> and <code>value</code> . Adjacent tables are used as the graph data structure, and outbound edges of a vertex are stored in edges of the vertex.
GraphLoader	<code>GraphLoader</code> is used to load graphs. <code>GraphLoader</code> is implemented by using the <code>setGraphLoaderClass</code> interface of <code>GraphJob</code> .
VertexResolver	<code>VertexResolver</code> is used to customize the conflict processing logic to modify graph topology. The <code>setLoadingVertexResolverClass</code> and <code>setComputingVertexResolverClass</code> interfaces of <code>GraphJob</code> provide the conflict processing logic for graph topology modification during graph loading and iteration calculation.
Partitioner	<code>Partitioner</code> is used to partition a graph so that the calculations can be fragmented. <code>Partitioner</code> is implemented by using the <code>setPartitionerClass</code> interface of <code>GraphJob</code> . <code>HashPartitioner</code> is used by default, that is, the hash value of a vertex ID is calculated and then a modulo operation is performed for the number of Workers.
WorkerComputer	<code>WorkerComputer</code> allows a Worker to run a custom logic during startup and exit. <code>WorkerComputer</code> is implemented by using the <code>setWorkerComputerClass</code> interface of a <code>GraphJob</code> .
Aggregator	<code>setAggregatorClass(Class ...)</code> defines one or multiple Aggregators.
Combiner	<code>setCombinerClass</code> sets a Combiner.
Counters	Indicates a counter. In job running logic, the <code>WorkerContext</code> interface can be used to obtain counters and perform counting. The framework automatically sums up the result.
WorkerContext	Indicates the context object. It encapsulates functions provided by the framework, such as modifying a graph topology, sending a message, writing a result, and reading a resource.

9.6. Development and debugging

This topic describes how to use Eclipse to develop MaxCompute Graph programs because Graph development plug-ins are unavailable in MaxCompute.

Graph development Graph debugging

The development process is as follows:

1. Compile Graph code and perform local debugging to test basic functions.
2. Perform cluster debugging to verify the result.

Development example

This topic uses the **SSSP** algorithm as an example to describe how to use Eclipse to develop and debug a MaxCompute Graph program.

Procedure:

1. Create a Java project named *graph_examples*.
2. Add the JAR package in the *lib* directory on the MaxCompute client to Java Build Path of the Eclipse project.

□

3. Develop a MaxCompute Graph program.

In the actual development process, an example program, such as **SSSP**, is first copied and then modified. In this example, only the package path is changed to *package com.aliyun.odps.graph.example*.

4. Compile and package the code.

In an Eclipse environment, right-click the source code directory, namely, the *src* directory, and choose **Export > Java > JAR file** to generate a JAR package. Select the path to store the JAR package, such as *D:\odps\clt\odps-graph-example-sssp.jar*.

5. Run SSSP on the MaxCompute client. For more information, see [\(Optional\) Submit Graph jobs](#).

Local debugging

MaxCompute Graph supports the local debugging mode. You can use Eclipse for breakpoint debugging.

Procedure:

1. Download a Maven package named *odps-graph-local*.
2. Select the Eclipse project, right-click the main program file that contains the `main` function of a Graph job, and choose **Run As > Run Configurations** to set parameters.
3. On the **Arguments** tab, set Program arguments to *1 sssp_in sssp_out* as the input parameter of the main program.
4. On the **Arguments** tab, set VM arguments to the following content:

```
-Dodps.runner.mode=local
-Dodps.project.name=<project.name>
-Dodps.end.point=<end.point>
-Dodps.access.id=<access.id>
-Dodps.access.key=<access.key>
```

5. In local mode in which *odps.end.point* is not specified, create the *sssp_in* and *sssp_out* tables in the *warehouse* directory and add the following data to the *sssp_in* table:

```
1,"2:2,3:1,4:4"
2,"1:2,3:2,4:1"
3,"1:1,2:2,5:1"
4,"1:4,2:1,5:1"
5,"3:1,4:1"
```

For more information about the *warehouse* directory, see [Local run](#).

6. Click Run to run SSSP on the local client.

-  **Note** Configure the parameters based on the settings in *conf/odps_config.ini* on the MaxCompute client. The preceding parameters are commonly used. Other parameters are described as follows:
- `odps.runner.mode`: The value is *local*. This parameter is required for the local debugging feature.
 - `odps.project.name`: specifies the current project. This parameter is required.
 - `odps.end.point`: specifies the endpoint of MaxCompute, which is optional. If this parameter is not specified, metadata and data of tables or resources are only read from the *warehouse* directory. An exception is reported if such data does not exist in the directory. If this parameter is specified, metadata and data are read from the *warehouse* directory first and then from the remote MaxCompute server if such data does not exist in the directory.
 - `odps.access.id`: specifies the AccessKey ID used to access MaxCompute. This parameter is valid only if `odps.end.point` is specified.
 - `odps.access.key`: specifies the AccessKey secret used to access MaxCompute. This parameter is valid only if `odps.end.point` is specified.
 - `odps.cache.resources`: specifies the resources you want to use. This parameter functions the same as `-resources` of the jar command.
 - `odps.local.warehouse`: specifies the local path to *warehouse*. The default value is *./warehouse*.

The output of the local SSSP debugging in Eclipse is as follows:

```
Counters: 3
com.aliyun.odps.graph.local.COUNTER
TASK_INPUT_BYTE=211
TASK_INPUT_RECORD=5
TASK_OUTPUT_BYTE=161
TASK_OUTPUT_RECORD=5
graph task finish
```

-  **Note** In this example, the *sssp_in* and *sssp_out* tables must exist in the local *warehouse* directory. For more information about the *sssp_in* and *sssp_out* tables, see [\(Optional\) Submit Graph jobs](#).

Temporary directory of a local job

A temporary directory is created in the Eclipse project directory each time local debugging is performed.

□

The temporary directory of a local Graph job contains the following directories and files:

- *counters*: stores counter information that is generated during the job running.
- *inputs*: stores input data of the job. Data is read from the local *warehouse* directory first. If no data is available, the MaxCompute SDK reads data from the server if `odps.end.point` is specified. An *input* directory reads only 10 data records by default. This threshold can be changed by using the `-Dodps.mapred.local.record.limit` parameter, of which the maximum value is 10000.
- *outputs*: stores output data of the job. If an output table exists in the local *warehouse* directory, the results in *outputs* will overwrite data in that table after the job is complete.
- *resources*: stores resources used by the job. Similar to input data, resource data is read from the local *warehouse* directory first. If no data is available, the MaxCompute SDK reads data from the server if `odps.end.point` is specified.
- *job.xml*: stores job configurations.
- *superstep*: stores persistent messages from each iteration.

 **Note** If detailed logs must be recorded during local debugging, you must create a *log4j* configuration file named *log4j.properties_odps_graph_cluster_debug* in the *src* directory.

Cluster debugging

After local debugging, you can submit the job to a cluster for testing.

Procedure:

1. Configure the MaxCompute client.
2. Run the `add jar /path/work.jar -f;` command to update the JAR package.
3. Run a jar command to execute the job and check the operational log and command output.

 **Note** For more information about how to run a Graph job in a cluster, see [\(Optional\) Submit Graph jobs](#).

Performance tuning

The configurations that affect the Graph performance are as follows:

- `setSplitSize(long)` : specifies the split size of an input table. The unit is MB. The value must be greater than 0. The default value is 64.
- `setNumWorkers(int)` : specifies the number of workers for a job. The value ranges from 1 to 1000. The default value is 1. The number of workers is determined by the input bytes of the job and `splitSize` .
- `setWorkerCPU(int)` : specifies CPU resources of the Map. The value ranges from 50 to 800. The default value is 200. A one-core CPU contains 100 resources.
- `setWorkerMemory(int)` : specifies memory resources of the Map. The unit is MB. The memory ranges from 256 MB to 12 GB. The default value is 4096 MB.
- `setMaxIteration(int)` : specifies the maximum number of iterations. The default value is -1. If the value is less than or equal to 0, the job does not stop if the maximum number of iterations is reached.

- `setJobPriority(int)` : specifies the job priority. The value ranges from 0 to 9. The default value is 9. A greater value indicates a lower priority.

We recommend that you optimize the performance by using one or more of the following methods:

- Use `setNumWorkers` to increase the number of workers.
- Use `setSplitSize` to reduce the split size and increase the data loading speed.
- Increase the CPU or memory resources for workers.
- Set the maximum number of iterations. For applications that do not require precise results, you can reduce the number of iterations to accelerate the execution process.

`setNumWorkers` and `setSplitSize` can be used together to accelerate data loading. Assume that the value of `setNumWorkers` equals that of `workerNum`, the value of `setSplitSize` equals that of `splitSize`, and the total number of input bytes is the value of `inputSize`. The number of split data records is calculated by using the following formula:

```
splitNum = inputSize / splitSize
```

The relationship between `workerNum` and `splitNum` is as follows:

- If the value of `splitNum` is equal to that of `workerNum`, each worker loads one split data record.
- If the value of `splitNum` is greater than that of `workerNum`, each worker loads one or more split data records.
- If the value of `splitNum` is less than that of `workerNum`, each worker loads up to one split data record.

Therefore, you can adjust `workerNum` and `splitSize` to obtain a suitable data loading speed. In the first two cases, data is loaded faster. In the iteration phase, you only need to adjust `workerNum`. If you set `runtime partitioning` to `False`, we recommend that you either use `setSplitSize` to adjust the number of workers or ensure the conditions in the first two cases are met. In the third case, some workers do not load split data records. Therefore, you can insert `set odps.graph.split.size=<m>; set odps.graph.worker.num=<n>;` before the jar command to achieve the same effect as `setNumWorkers` and `setSplitSize`.

Another common performance issue is data skew. As indicated by the counters, some workers process excessive split data records or edges than others. Data skew occurs when the number of split data records, edges, or messages that correspond to some keys is much greater than other keys. These keys are processed by a small number of workers, which results in longer runtimes.

To address this issue, try one or more of the following methods:

- Use a combiner to aggregate the messages of split data records that correspond to the keys to reduce the number of generated messages.

Developers can define a combiner to reduce the memory and network traffic consumed by message storage, which reduces the job execution duration.

- Improve the business logic.

When the data volume is large, reading data in a disk may take up the processing time. Therefore, you can reduce the data bytes to be read to increase the overall throughput. This improves job performance. To reduce data bytes, use one of the following methods:

- Reduce data input: For some decision-making applications, processing sampled data only affects the precision of the results, not the overall accuracy. In this case, you can perform special data sampling and import the data to the input table for processing.
- Avoid reading fields that are not used: The `TableInfo` class of the MaxCompute Graph framework supports reading specific columns that are transferred by using column name arrays, rather than reading the entire table or partition. This reduces the input data volume and improves job performance.

Built-in JAR packages

By default, the following JAR packages are loaded on a JVM that runs Graph programs. You do not need to manually upload these resources or use `-libjars` to specify them in a command:

- *commons-codec-1.3.jar*
- *commons-io-2.0.1.jar*
- *commons-lang-2.5.jar*
- *commons-logging-1.0.4.jar*
- *commons-logging-api-1.0.4.jar*
- *guava-14.0.jar*
- *json.jar*
- *log4j-1.2.15.jar*
- *slf4j-api-1.4.3.jar*
- *slf4j-log4j12-1.4.3.jar*
- *xmlenc-0.52.jar*

 **Note** In the classpath of the JVM, the preceding built-in JAR packages are placed before your JAR packages, which may result in a version conflict. For example, your program calls a certain class function in *commons-codec-1.5.jar*, but the function is not included in *commons-codec-1.3.jar*. In this case, you can choose to call a similar function in *commons-codec-1.3.jar* or wait until MaxCompute is upgraded to the required version.

9.7. Examples

9.7.1. SSSP

Dijkstra is a typical algorithm that calculates the Single Source Shortest Path (SSSP) in a directed graph.

For weighted directed graph $G=(V,E)$, many paths are routed from source vertex s to sink vertex v . In these paths, the one that has the smallest edge weight sum is called the shortest distance from s to v .

The basic concept of the algorithm is as follows:

- Initialization: The distance from source vertex s to s itself is zero ($d[s] = 0$), and the distance

from another vertex u to s is infinite ($d[u]=\infty$).

- Iteration: If an edge exists from u to v , the shortest distance from s to v is updated as: $d[v] = \min(d[v], d[u] + \text{weight}(u, v))$. The iteration ends until the distance from all vertices to s does not change.

The basic concept shows that the algorithm is applicable to solutions using the MaxCompute Graph program. Each vertex maintains the current shortest distance to the source vertex. If the value changes, a message containing the new value and the edge weight is sent to the adjacent vertex. In the next iteration, the adjacent vertex updates the current shortest distance based on the received message. The iteration ends when the current shortest distance of all vertices does not change.

Sample Code

Code of SSSP is as follows:

```
import java.io.IOException;

import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.Combiner;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.data.TableInfo;

public class SSSP {

    public static final String START_VERTEX = "sssp.start.vertex.id";

    public static class SSSPVertex extends
        Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {

        private static long startVertexId = -1;

        public SSSPVertex() {
            this.setValue(new LongWritable(Long.MAX_VALUE));
        }

        public boolean isStartVertex(
            ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context) {
```

```

    if (startVertexId == -1) {
        String s = context.getConfiguration().get(START_VERTEX);
        startVertexId = Long.parseLong(s);

    return getId().get() == startVertexId;

@Override
public void compute(
    ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context,
    Iterable<LongWritable> messages) throws IOException {
    long minDist = isStartVertex(context) ? 0 : Integer.MAX_VALUE;
    for (LongWritable msg : messages) {
        if (msg.get() < minDist) {
            minDist = msg.get();

    if (minDist < this.getValue().get()) {
        this.setValue(new LongWritable(minDist));
        if (hasEdges()) {
            for (Edge<LongWritable, LongWritable> e : this.getEdges()) {
                context.sendMessage(e.getDestVertexId(), new LongWritable(minDist
                    + e.getValue().get()));

    } else {
        voteToHalt();

@Override
public void cleanup(
    WorkerContext<LongWritable, LongWritable, LongWritable, LongWritable> context)
    throws IOException {
    context.write(getId(), getValue());

public static class MinLongCombiner extends
    Combiner<LongWritable, LongWritable> {

```

```

    combine<LongWritable, LongWritable> {

    @Override
    public void combine(LongWritable vertexId, LongWritable combinedMessage,
        LongWritable messageToCombine) throws IOException {
        if (combinedMessage.get() > messageToCombine.get()) {
            combinedMessage.set(messageToCombine.get());
        }
    }

    public static class SSSPVertexReader extends
        GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {

        @Override
        public void load(
            LongWritable recordNum,
            WritableRecord record,
            MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context)
            throws IOException {
            SSSPVertex vertex = new SSSPVertex();
            vertex.setId((LongWritable) record.get(0));
            String[] edges = record.get(1).toString().split(",");
            for (int i = 0; i < edges.length; i++) {
                String[] ss = edges[i].split(":");
                vertex.addEdge(new LongWritable(Long.parseLong(ss[0])),
                    new LongWritable(Long.parseLong(ss[1])));
            }

            context.addVertexRequest(vertex);
        }

        public static void main(String[] args) throws IOException {
            if (args.length < 2) {
                System.out.println("Usage: <startnode> <input> <output>");
                System.exit(-1);
            }
        }
    }
}

```

```

GraphJob job = new GraphJob();
job.setGraphLoaderClass(SSSPVertexReader.class);
job.setVertexClass(SSSPVertex.class);
job.setCombinerClass(MinLongCombiner.class);

job.set(START_VERTEX, args[0]);
job.addInput(TableInfo.builder().tableName(args[1]).build());
job.addOutput(TableInfo.builder().tableName(args[2]).build());

long startTime = System.currentTimeMillis();
job.run();
System.out.println("Job Finished in "
    + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");

```

The source code of SSSP is described as follows:

- Row 19: Defines SSSPVertex, where:
 - The vertex value indicates the current shortest distance from this vertex to source vertex startVertexId.
 - The compute() method uses the iteration formula $d[v] = \min(d[v], d[u] + \text{weight}(u, v))$ to update the vertex value.
 - The cleanup() method writes the vertex and its shortest distance to the source vertex to the result table.
- Row 58: If the vertex value does not change, voteToHalt() is called to notify the framework that this vertex enters the halt status. The calculation ends when all vertices enter the halt state.
- Row 70: Defines MinLongCombiner and combines messages sent to the same vertex to optimize performance and reduce memory usage.
- Row 83: Defines the SSSPVertexReader class, loads a graph, and resolves each record in the table into a vertex. The first column of the record is the vertex ID, and the second column stores all edge sets starting from the vertex, such as 2:2, 3:1, 4:4.
- Row 106: Runs the main program (main function), defines GraphJob, and specifies the implementation of Vertex/GraphLoader/Combiner, and the input and output tables.

9.7.2. PageRank

PageRank is a typical algorithm used to calculate the web page ranking. In the input directed graph G , vertices indicate web pages. If a link exists between web pages A and B, an edge connecting A and B exists.

The basic concept of the algorithm is as follows:

- Initialization: The vertex value indicates the rank value (of the double type) of PageRank. In

the initial phase, the value of all vertices is $1/\text{TotalNumVertices}$.

- Iteration formula: $\text{PageRank}(i) = 0.15/\text{TotalNumVertices} + 0.85 \times \text{sum}$. Sum indicates the sum of $\text{PageRank}(j)/\text{out\_degree}(j)$. (j indicates all vertices pointing to vertex i.)

The basic concept shows that the algorithm is applicable to solutions using the MaxCompute Graph program. Each vertex j maintains the value of PageRank. $\text{PageRank}(j)/\text{out\_degree}(j)$ is sent to the adjacent vertex (for voting) in each iteration. In the next iteration, each vertex recalculates the PageRank value using the iteration formula.

Sample Code

```
import java.io.IOException;

import org.apache.log4j.Logger;

import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Writable;

public class PageRank {

    private final static Logger LOG = Logger.getLogger(PageRank.class);

    public static class PageRankVertex extends
        Vertex<Text, DoubleWritable, NullWritable, DoubleWritable> {

        @Override
        public void compute(
            ComputeContext<Text, DoubleWritable, NullWritable, DoubleWritable> context,
            Iterable<DoubleWritable> messages) throws IOException {
            if (context.getSuperstep() == 0) {
                setValue(new DoubleWritable(1.0 / context.getTotalNumVertices()));
            } else if (context.getSuperstep() >= 1) {
                ...
            }
        }
    }
}
```



```

    double sum = 0;
    for (DoubleWritable msg : messages) {
        sum += msg.get();

        DoubleWritable vertexValue = new DoubleWritable(
            (0.15f / context.getTotalNumVertices()) + 0.85f * sum);
        setValue(vertexValue);

        if (hasEdges()) {
            context.sendMessageToNeighbors(this, new DoubleWritable(getValue()
                .get() / getEdges().size()));
        }
    }

    @Override
    public void cleanup(
        WorkerContext<Text, DoubleWritable, NullWritable, DoubleWritable> context)
        throws IOException {
        context.write(getId(), getValue());
    }

    public static class PageRankVertexReader extends
        GraphLoader<Text, DoubleWritable, NullWritable, DoubleWritable> {

        @Override
        public void load(
            LongWritable recordNum,
            WritableRecord record,
            MutationContext<Text, DoubleWritable, NullWritable, DoubleWritable> context)
            throws IOException {
            PageRankVertex vertex = new PageRankVertex();
            vertex.setValue(new DoubleWritable(0));
            vertex.setId((Text) record.get(0));
            System.out.println(record.get(0));

            for (int i = 1; i < record.size(); i++) {
                Writable edge = record.get(i);
                System.out.println(edge.toString());
                if (!(edge.equals(NullWritable.get()))) {
                    vertex.addEdge(new Text(edge.toString()), NullWritable.get());
                }
            }
        }
    }

```

```
LOG.info("vertex eds size: "
    + (vertex.hasEdges() ? vertex.getEdges().size() : 0));
context.addVertexRequest(vertex);

private static void printUsage() {
    System.out.println("Usage: <in> <out> [Max iterations (default 30)]");
    System.exit(-1);

public static void main(String[] args) throws IOException {
    if (args.length < 2)
        printUsage();

    GraphJob job = new GraphJob();

    job.setGraphLoaderClass(PageRankVertexReader.class);
    job.setVertexClass(PageRankVertex.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());

    // default max iteration is 30
    job.setMaxIteration(30);
    if (args.length >= 3)
        job.setMaxIteration(Integer.parseInt(args[2]));

    long startTime = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
```

The source code of PageRank is described as follows:

- Row 23: Defines PageRankVertex, where:
 - The vertex value indicates the current PageRank value of the vertex (web page).

- The `compute()` method uses the iteration formula $\text{PageRank}(i) = 0.15/\text{TotalNumVertices} + 0.85 \times \text{sum}$ to update the vertex value.
- The `cleanup()` method writes the vertex and its PageRank value to the result table.
- Row 55: Defines the `PageRankVertexReader` class, loads a graph, and resolves each record in the table into a vertex. The first column of the record is the start vertex and other columns are the destination vertices.
- Row 88: Runs the main program (main function), defines `GraphJob`, and specifies the implementation of `Vertex/GraphLoader`, the maximum number of iterations (30 by default), and input and output tables.

9.7.3. Kmeans

The Kmeans algorithm is a typical clustering algorithm.

It performs clustering by using k number of vertices in the space as the centers and grouping the vertices closest to them. The values of the clustering centers are successively updated through iterations until the optimal clustering result is obtained.

To divide a sample set into k classes, the algorithm operates as follows:

1. Selects the initial centers of k classes.
2. Calculates the distance from any sample to the k centers in iteration i, and groups the sample to the class of the nearest center.
3. Updates the center value of the class using the mean and other methods.
4. For all k clustering centers, if the value updated after iterations remains unchanged or is smaller than a threshold, the iteration ends. Otherwise, the iteration continues.

Sample Code

Code for the K-means clustering algorithm is as follows:

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.log4j.Logger;

import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.ODPS.graph.computercontext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
```

```
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;

public class Kmeans {
    private final static Logger LOG = Logger.getLogger(Kmeans.class);

    public static class KmeansVertex extends
        Vertex<Text, Tuple, NullWritable, NullWritable> {

        @Override
        public void compute(
            ComputeContext<Text, Tuple, NullWritable, NullWritable> context,
            Iterable<NullWritable> messages) throws IOException {
            context.aggregate(getValue());
        }

        public static class KmeansVertexReader extends
            GraphLoader<Text, Tuple, NullWritable, NullWritable> {
                @Override
                public void load(LongWritable recordNum, WritableRecord record,
                    MutationContext<Text, Tuple, NullWritable, NullWritable> context)
                    throws IOException {
                    KmeansVertex vertex = new KmeansVertex();
                    vertex.setId(new Text(String.valueOf(recordNum.get())));
                    vertex.setValue(new Tuple(record.getAll()));
                    context.addVertexRequest(vertex);
                }
            }

        public static class KmeansAggrValue implements Writable {

            Tuple centers = new Tuple();
            Tuple sums = new Tuple();
            Tuple counts = new Tuple();
        }
    }
}
```

```

@ Override
public void write(DataOutput out) throws IOException {
    centers.write(out);
    sums.write(out);
    counts.write(out);

@ Override
public void readFields(DataInput in) throws IOException {
    centers = new Tuple();
    centers.readFields(in);
    sums = new Tuple();
    sums.readFields(in);
    counts = new Tuple();
    counts.readFields(in);

@ Override
public String toString() {
    return "centers " + centers.toString() + ", sums " + sums.toString()
        + ", counts " + counts.toString();

public static class KmeansAggregator extends Aggregator<KmeansAggrValue> {

@ SuppressWarnings("rawtypes")
@ Override
public KmeansAggrValue createInitialValue(WorkerContext context)
    throws IOException {
    KmeansAggrValue aggrVal = null;
    if (context.getSuperstep() == 0) {
        aggrVal = new KmeansAggrValue();
        aggrVal.centers = new Tuple();
        aggrVal.sums = new Tuple();
        aggrVal.counts = new Tuple();

        byte[] centers = context.readCacheFile("centers");
        String lines[] = new String(centers).split("\n");

```

```
for (int i = 0; i < lines.length; i++) {
    String[] ss = lines[i].split(",");
    Tuple center = new Tuple();
    Tuple sum = new Tuple();
    for (int j = 0; j < ss.length; ++j) {
        center.append(new DoubleWritable(Double.valueOf(ss[j].trim())));
        sum.append(new DoubleWritable(0.0));
    }

    LongWritable count = new LongWritable(0);
    aggrVal.sums.append(sum);
    aggrVal.counts.append(count);
    aggrVal.centers.append(center);

} else {
    aggrVal = (KmeansAggrValue) context.getLastAggregatedValue(0);

return aggrVal;

@Override
Public void aggregate (glasvalue, object item ){
    int min = 0;
    double mindist = Double.MAX_VALUE;
    Tuple point = (Tuple) item;

    for (int i = 0; i < value.centers.size(); i++) {
        Tuple center = (Tuple) value.centers.get(i);
        // use Euclidean Distance, no need to calculate sqrt
        double dist = 0.0d;
        for (int j = 0; j < center.size(); j++) {
            double v = ((DoubleWritable) point.get(j)).get()
                - ((DoubleWritable) center.get(j)).get();
            dist += v * v;
        }

        if (dist < mindist) {
            mindist = dist;
            min = i;
        }
    }
}
```

```

// update sum and count
Tuple sum = (Tuple) value.sums.get(min);
for (int i = 0; i < point.size(); i++) {
    DoubleWritable s = (DoubleWritable) sum.get(i);
    s.set(s.get() + ((DoubleWritable) point.get(i)).get());

    LongWritable count = (LongWritable) value.counts.get(min);
    count.set(count.get() + 1);
}

@Override
public void merge(KmeansAggrValue value, KmeansAggrValue partial) {
    for (int i = 0; i < value.sums.size(); i++) {
        Tuple sum = (Tuple) value.sums.get(i);
        Tuple that = (Tuple) partial.sums.get(i);
        for (int j = 0; j < sum.size(); j++) {
            DoubleWritable s = (DoubleWritable) sum.get(j);
            s.set(s.get() + ((DoubleWritable) that.get(j)).get());
        }
    }

    for (int i = 0; i < value.counts.size(); i++) {
        LongWritable count = (LongWritable) value.counts.get(i);
        count.set(count.get() + ((LongWritable) partial.counts.get(i)).get());
    }
}

@SuppressWarnings("rawtypes")
@Override
public boolean terminate(WorkerContext context, KmeansAggrValue value)
    throws IOException {
    // compute new centers
    Tuple newCenters = new Tuple(value.sums.size());
    for (int i = 0; i < value.sums.size(); i++) {
        Tuple sum = (Tuple) value.sums.get(i);
        Tuple newCenter = new Tuple(sum.size());
        LongWritable c = (LongWritable) value.counts.get(i);
        for (int j = 0; j < sum.size(); j++) {

```

```
DoubleWritable s = (DoubleWritable) sum.get(j);
double val = s.get() / c.get();
newCenter.set(j, new DoubleWritable(val));

// reset sum for next iteration
s.set(0.0d);

// reset count for next iteration
c.set(0);
newCenters.set(i, newCenter);

// update centers
Tuple oldCenters = value.centers;
value.centers = newCenters;

LOG.info("old centers: " + oldCenters + ", new centers: " + newCenters);

// compare new/old centers
boolean converged = true;
for (int i = 0; i < value.centers.size() && converged; i++) {
    Tuple oldCenter = (Tuple) oldCenters.get(i);
    Tuple newCenter = (Tuple) newCenters.get(i);
    double sum = 0.0d;
    for (int j = 0; j < newCenter.size(); j++) {
        double v = ((DoubleWritable) newCenter.get(j)).get()
            - ((DoubleWritable) oldCenter.get(j)).get();
        sum += v * v;
    }

    double dist = Math.sqrt(sum);
    LOG.info("old center: " + oldCenter + ", new center: " + newCenter
        + ", dist: " + dist);
    // converge threshold for each center: 0.05
    converged = dist < 0.05d;
}

if (converged || context.getSuperstep() == context.getMaxIteration() - 1) {
    // converged or reach max iteration, output centers
    for (int i = 0; i < value.centers.size(); i++) {
        context.write(((Tuple) value.centers.get(i)).toArray());
    }
}
```



```
// true means to terminate iteration
return true;

// false means to continue iteration
return false;

private static void printUsage() {
System.out.println ("Usage: <in> <out> [Max iterations (default 30)] ");
System.exit(-1);

public static void main(String[] args) throws IOException {
if (args.length < 2)
    printUsage();

    GraphJob job = new GraphJob();

    job.setGraphLoaderClass(KmeansVertexReader.class);
    job.setRuntimePartitioning(false);
    job.setVertexClass(KmeansVertex.class);
    job.setAggregatorClass(KmeansAggregator.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());

    // default max iteration is 30
    job.setMaxIteration(30);
    if (args.length >= 3)
        job.setMaxIteration(Integer.parseInt(args[2]));

    long start = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
```

The source code of Kmeans is described as follows:

- Row 26: Defines KmeansVertex. The compute() method is simple. It calls the aggregate() method of the context object and transmits the value of the current vertex (in Tuple type and expressed by vector).
- Row 38: Defines the KmeansVertexReader class, loads a graph, and resolves each record in the table as a vertex. The vertex ID does not matter, and transmitted recordNum is used as the ID. The vertex value is the Tuple consisting of all columns of the record.
- Row 83: Defines KmeansAggregator. This class encapsulates the main logic of the Kmeans algorithm, where:
 - createInitialValue creates an initial value for each iteration (k-class center point). In first iteration (superstep equals to 0), the value is the initial center point. Otherwise, the value is the new center point when the last iteration ends.
 - The aggregate() method calculates the distance from each vertex to centers of different classes, classifies the vertex as the class of the nearest center, and updates sum and count of the class.
 - The merge() method combines sums and counts collected by each Worker.
 - The terminate() method calculates the new center point based on sum and count of each class. If the distance between the new and old center points is smaller than a threshold value or the number of iterations reaches the upper limit, the iteration ends (false is returned). The final center point is written to the result table.
- Row 236: Runs the main program (main function), defines GraphJob, and specifies the implementation of Vertex/GraphLoader/Aggregator, the maximum number of iterations (30 by default), and the input and output tables.
- Row 243: Specifies job.setRuntimePartitioning(false). For the Kmeans algorithm, vertices do not have to be distributed during graph loading. If RuntimePartitioning is set to false, the performance for graph loading is improved.

9.7.4. BiPartiteMatchiing

A Bipartite graph means all the graph vertices can be separated into 2 sets, and 2 vertices corresponding to each Edge belong to the 2 sets respectively. For bipartite graph G, M is one of its sub-graphs. If any two edges in the edge set of M are not attached to the same vertex, M is called a matching. The bipartite graph matching is usually used for information matching in scenarios with clear supply and demand relationships.

The basic concept of the algorithm is as follows:

- From the first vertex on the left, unmatched vertices are selected to search for the augmenting path.
- If an unmatched vertex is found, the search is successful.
- The path information is updated. If the number of matching edges is increased by 1, the search is stopped.
- If the augmenting path is not found, the search is no longer started from this vertex.

Sample Code

BiPartiteMatchiing The code of the algorithm is as follows:

```
import java.io.DataInput;
```

```
import java.io.DataOutput;
import java.io.IOException;
import java.util.Random;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
public class BipartiteMatching {
    private static final Text UNMATCHED = new Text("UNMATCHED");
    public static class TextPair implements Writable {
        public Text first;
        public Text second;
        public TextPair() {
            first = new Text();
            second = new Text();
        }
        public TextPair(Text first, Text second) {
            this.first = new Text(first);
            this.second = new Text(second);
        }
        @ Override
        public void write(DataOutput out) throws IOException {
            first.write(out);
            second.write(out);
        }
        @ Override
        public void readFields(DataInput in) throws IOException {
            first = new Text();
            first.readFields(in);
            second = new Text();
            second.readFields(in);
        }
        @ Override
```

```

public String toString() {
    return first + ": " + second;
}

public static class BipartiteMatchingVertexReader extends
    GraphLoader<Text, TextPair, NullWritable, Text> {
    @ Override
    public void load(LongWritable recordNum, WritableRecord record,
        MutationContext<Text, TextPair, NullWritable, Text> context)
        throws IOException {
        BipartiteMatchingVertex vertex = new BipartiteMatchingVertex();
        vertex.setId((Text) record.get(0));
        vertex.setValue(new TextPair(UNMATCHED, (Text) record.get(1)));
        String[] adjs = record.get(2).toString().split(",");
        for (String adj : adjs) {
            vertex.addEdge(new Text(adj), null);
        }

        context.addVertexRequest(vertex);
    }
}

public static class BipartiteMatchingVertex extends
    Vertex<Text, TextPair, NullWritable, Text> {
    private static final Text LEFT = new Text("LEFT");
    private static final Text RIGHT = new Text("RIGHT");
    private static Random rand = new Random();
    @ Override
    public void compute (
        ComputeContext<Text, TextPair, NullWritable, Text> context,
        Iterable messages) throws IOException {
        if (isMatched()) {
            voteToHalt();
            return;
        }

        switch ((int) context.getSuperstep() % 4) {
        case 0:
            if (isLeft()) {
                context.sendMessageToNeighbors(this, getId());
            }

            break;
        case 1:
            if (isRight()) {
                context.sendMessageToNeighbors(this, getId());
            }

            break;
        case 2:
            if (isLeft()) {
                context.sendMessageToNeighbors(this, getId());
            }

            break;
        case 3:
            if (isRight()) {
                context.sendMessageToNeighbors(this, getId());
            }

            break;
        }
    }
}

```

```
if (isRight()) {
    Text luckyLeft = null;
    for (Text message : messages) {
        if (luckyLeft == null) {
            luckyLeft = new Text(message);
        } else {
            if (rand.nextInt(1) == 0) {
                luckyLeft.set(message);
            }
        }

        if (luckyLeft != null) {
            context.sendMessage(luckyLeft, getId());
        }

        break;
    }
}

case 2:
    if (isLeft()) {
        Text luckyRight = null;
        for (Text msg : messages) {
            if (luckyRight == null) {
                luckyRight = new Text(msg);
            } else {
                if (rand.nextInt(1) == 0) {
                    luckyRight.set(msg);
                }
            }

            if (luckyRight != null) {
                setMatchVertex(luckyRight);
                context.sendMessage(luckyRight, getId());
            }

            break;
        }
    }

case 3:
    if (isRight()) {
        for (Text msg : messages) {
            setMatchVertex(msg);
        }

        break;
    }
```

```
@ Override
public void cleanup(
    WorkerContext<Text, TextPair, NullWritable, Text> context)
    throws IOException {
    context.write(getId(), getValue().first);

    private boolean isMatched() {
        return ! getValue().first.equals(UNMATCHED);

    private boolean isLeft() {
        return getValue().second.equals(LEFT);

    private boolean isRight() {
        return getValue().second.equals(RIGHT);

    private void setMatchVertex(Text matchVertex) {
        getValue().first.set(matchVertex);

    private static void printUsage() {
        System.err.println("BipartiteMatching <input> <output> [maxIteration]");

    public static void main(String[] args) throws IOException {
        if (args.length < 2) {
            printUsage();

        GraphJob job = new GraphJob();
        job.setGraphLoaderClass(BipartiteMatchingVertexReader.class);
        job.setVertexClass(BipartiteMatchingVertex.class);
        job.addInput(TableInfo.builder().tableName(args[0]).build());
        job.addOutput(TableInfo.builder().tableName(args[1]).build());
        int maxIteration = 30;
        if (args.length > 2) {
            maxIteration = Integer.parseInt(args[2]);

        job.setMaxIteration(maxIteration);
        job.run();
```

9.7.5. Strongly-connected component

In a digraph, if by starting from any vertex it reaches every vertex in the graph through Edges, it is called a strongly-connected graph. A strongly-connected subgraph with an extremely large vertex number is called a strongly-connected component. The algorithm is based on [Parallel coloring algorithm](#).

Each vertex contains the following components:

- **colorID:** Stores the color of vertex *v* during forward traversal. After a calculation ends, vertices with the same colorID belong to one strongly connected component.
- **transposeNeighbors:** Stores the neighbor ID of vertex *v* in the transpose graph of the input graph.

The algorithm contains the following four steps:

- **Transpose graph generation:** Contains two supersteps. Each vertex sends its ID to its neighbor with the corresponding outbound edge. In the next superstep, these IDs are stored as transposeNeighbors values.
- **Trim:** Contains one superstep. Each vertex that has only one inbound or outbound edge sets the colorID as its own ID and the status to inactive. Subsequent signals sent to the vertex are ignored.
- **Forward traversal:** A vertex contains two sub-processes (supersteps): startup and sleep. In the startup phase, each vertex sets the colorID as its own ID and sends the ID to the neighbor with the corresponding outbound edge. In the sleep phase, the vertex uses the maximum colorID it received to update its own colorID, and transmits the colorID until the colorID converges. When the colorID converges, the master process sets the global object to backward traversal.
- **Backward traversal:** Contains two sub-processes: startup and sleep. In the startup phase, a vertex whose ID is the same as the colorID transmits its ID to the neighbor vertex in the transpose graph, and sets its status to inactive. Subsequent signals sent to the vertex can be ignored. In each sleep step, each vertex receives signals matching its colorID, transmits the colorID in the transpose graph, and then sets its status to inactive. If active vertices exist after this step ends, the process reverts to the trim step.

Sample Code

The code for strongly connected components is as follows:

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
```

```
import com.aliyun.odps.graph.Vertex;  
import com.aliyun.odps.graph.WorkerContext;  
import com.aliyun.odps.io.BooleanWritable;  
import com.aliyun.odps.io.IntWritable;  
import com.aliyun.odps.io.LongWritable;  
import com.aliyun.odps.io.NullWritable;  
import com.aliyun.odps.io.Tuple;  
import com.aliyun.odps.io.Writable;  
import com.aliyun.odps.io.WritableRecord;
```

* Definition from Wikipedia:

* In the mathematical theory of directed graphs, a graph is said
* to be strongly connected if every vertex is reachable from every
* other vertex. The strongly connected components of an arbitrary
* directed graph form a partition into subgraphs that are themselves
* Strictly connected.

* Algorithms with four phases as follows.

* 1. Transpose Graph Formation: Requires two supersteps. In the first
* superstep, each vertex sends a message with its ID to all its outgoing
* neighbors, which in the second superstep are stored in transposeNeighbors.

* 2. Trimming: Takes one superstep. Every vertex with only in-coming or
* only outgoing edges (or neither) sets its colorID to its own ID and
* becomes inactive. Messages subsequently sent to the vertex are ignored.

* 3. Forward-Traversal: There are two sub phases: Start and Rest. In the
* Start phase, each vertex sets its colorID to its own ID and propagates
* its ID to its outgoing neighbors. In the Rest phase, vertices update
* their own colorIDs with the minimum colorID they have seen, and propagate
* their colorIDs, if updated, until the colorIDs converge.
* Set the phase to Backward-Traversal when the colorIDs converge.

* 4. Backward-Traversal: We again break the phase into Start and Rest.
* In Start, every vertex whose ID equals its colorID propagates its ID to
* the vertices in transposeNeighbors and sets itself inactive. Messages
* subsequently sent to the vertex are ignored. In each of the Rest phase supersteps,
* each vertex receiving a message that matches its colorID: (1) propagates
* its colorID in the transpose graph; (2) sets itself inactive. Messages
* subsequently sent to the vertex are ignored. Set the phase back to Trimming
* if not all vertex are inactive.

* <http://ilpubs.stanford.edu:8090/1077/3/p535-salihoglu.pdf>

```
public class StronglyConnectedComponents {  
    public final static int STAGE_TRANSPOSE_1 = 0;  
    public final static int STAGE_TRANSPOSE_2 = 1;  
    public final static int STAGE_TRIMMING = 2;  
    public final static int STAGE_FW_START = 3;  
    public final static int STAGE_FW_REST = 4;  
    public final static int STAGE_BW_START = 5;  
    public final static int STAGE_BW_REST = 6;
```

* The value is composed of component id, incoming neighbors,
* active status and updated status.

```
public static class MyValue implements Writable {  
    LongWritable sccID; // strongly connected component id  
    Tuple inNeighbors; // transpose neighbors  
    BooleanWritable active; // vertex is active or not  
    BooleanWritable updated; // sccID is updated or not  
    public MyValue() {  
        this.sccID = new LongWritable(Long.MAX_VALUE);  
        this.inNeighbors = new Tuple();  
        this.active = new BooleanWritable(true);  
        this.updated = new BooleanWritable(false);  
  
    public void setSccID(LongWritable sccID) {  
        this.sccID = sccID;  
  
    public LongWritable getSccID() {  
        return this.sccID;  
  
    public void setInNeighbors(Tuple inNeighbors) {  
        this.inNeighbors = inNeighbors;  
  
    public Tuple getInNeighbors() {  
        return this.inNeighbors;  
  
    public void addInNeighbor(LongWritable neighbor) {  
        this.inNeighbors.append(new LongWritable(neighbor.get()));
```

```
public boolean isActive() {
    return this.active.get();

public void setActive(boolean status) {
    this.active.set(status);

public boolean isUpdated() {
    return this.updated.get();

public void setUpdated(boolean update) {
    this.updated.set(update);

@Override
public void write(DataOutput out) throws IOException {
    this.sccID.write(out);
    this.inNeighbors.write(out);
    this.active.write(out);
    this.updated.write(out);

@Override
public void readFields(DataInput in) throws IOException {
    this.sccID.readFields(in);
    this.inNeighbors.readFields(in);
    this.active.readFields(in);
    this.updated.readFields(in);

@Override
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("sccID: " + sccID.get());
    sb.append(" inNeighbors: " + inNeighbors.toDelimitedString(','));
    sb.append(" active: " + active.get());
    sb.append(" updated: " + updated.get());
    return sb.toString();

public static class SCCVertex extends
Vertex <LongWritable, MyValue, NullWritable, LongWritable> {
    public SCCVertex() {
        this.setValue(new MyValue());
```

```

@Override
public void compute(
    ComputeContext < LongWritable, MyValue, NullWritable, LongWritable> context,
    Iterable <LongWritable> msgs) throws IOException {
    // Messages sent to inactive vertex are ignored.
    if (! this.getValue().isActive()) {
        this.voteToHalt();
        return;

    int stage = ((SCCAggrValue)context.getLastAggregatedValue(0)).getStage();
    switch (stage) {
    case STAGE_TRANSPOSE_1:
        context.sendMessageToNeighbors(this, this.getId());
        break;
    case STAGE_TRANSPOSE_2:
        for (LongWritable msg: msgs) {
            this.getValue().addInNeighbor(msg);

    case STAGE_TRIMMING:
        this.getValue().setScclD(getId());
        if (this.getValue().getInNeighbors().size() == 0 ||
            this.getNumEdges() == 0) {
            this.getValue().setActive(false);

        break;
    case STAGE_FW_START:
        this.getValue().setScclD(getId());
        context.sendMessageToNeighbors(this, this.getValue().getScclD());
        break;
    case STAGE_FW_REST:
        long minScclD = Long.MAX_VALUE;
        for (LongWritable msg : msgs) {
            if (msg.get() < minScclD) {
                minScclD = msg.get();

        if (minScclD < this.getValue().getScclD().get()) {
            this.getValue().setScclD(new LongWritable(minScclD));
            context.sendMessageToNeighbors(this, this.getValue().getScclD());
            this.getValue().setUpdated(true);

```

```

    } else {
        this.getValue().setUpdated(false);

        break;
    case STAGE_BW_START:
        if (this.getId().equals(this.getValue().getScclID())) {
            for (Writable neighbor : this.getValue().getInNeighbors().getAll()) {
                context.sendMessage((LongWritable)neighbor, this.getValue().getScclID());

                this.getValue().setActive(false);

                break;
            }
        case STAGE_BW_REST:
            this.getValue().setUpdated(false);
            for (LongWritable msg : msgs) {
                if (msg.equals(this.getValue().getScclID())) {
                    for (Writable neighbor : this.getValue().getInNeighbors().getAll()) {
                        context.sendMessage((LongWritable)neighbor, this.getValue().getScclID());

                        this.getValue().setActive(false);
                        this.getValue().setUpdated(true);
                        break;
                    }
                }
            }

            break;

            context.aggregate(0, getValue());

@Override
public void cleanup(
    WorkerContext<LongWritable, MyValue, NullWritable, LongWritable> context)
    throws IOException {
    context.write(getId(), getValue().getScclID());

    * The SCCAggrValue maintains global stage and graph updated and active status.
    * updated is true only if one vertex is updated.
    * active is true only if one vertex is active.

    public static class SCCAagrValue implements Writable {

```

```

    private static final Logger LOG = LoggerFactory.getLogger(SCCAggregator.class);

    private IntWritable stage = new IntWritable(STAGE_TRANSPOSE_1);
    private BooleanWritable updated = new BooleanWritable(false);
    private BooleanWritable active = new BooleanWritable(false);

    public void setStage(int stage) {
        this.stage.set(stage);
    }

    public int getStage() {
        return this.stage.get();
    }

    public void setUpdated(boolean updated) {
        this.updated.set(updated);
    }

    public boolean getUpdated() {
        return this.updated.get();
    }

    public void setActive(boolean active) {
        this.active.set(active);
    }

    public boolean getActive() {
        return this.active.get();
    }

    @Override
    public void write(DataOutput out) throws IOException {
        this.stage.write(out);
        this.updated.write(out);
        this.active.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        this.stage.readFields(in);
        this.updated.readFields(in);
        this.active.readFields(in);
    }

    * The job of SCCAggregator is to schedule global stage in every superstep.

    public static class SCCAggregator extends Aggregator<SCCAggrValue> {
        @SuppressWarnings("rawtypes")
        @Override

```

```
public SCCAggrValue createStartupValue(WorkerContext context) throws IOException {
    return new SCCAggrValue();

    @SuppressWarnings("rawtypes")
    @Override
    public SCCAggrValue createInitialValue(WorkerContext context)
        throws IOException {
        return (SCCAggrValue) context.getLastAggregatedValue(0);

    @Override
    public void aggregate(SCCAggrValue value, Object item) throws IOException {
        MyValue v = (MyValue)item;
        if ((value.getStage() == STAGE_FW_REST || value.getStage() == STAGE_BW_REST)
            && v.isUpdated()) {
            value.setUpdated(true);

            // only active vertex invoke aggregate()
            value.setActive(true);

    @Override
    public void merge(SCCAggrValue value, SCCAggrValue partial)
        throws IOException {
        boolean updated = value.getUpdated() || partial.getUpdated();
        value.setUpdated(updated);
        boolean active = value.getActive() || partial.getActive();
        value.setActive(active);

    @SuppressWarnings("rawtypes")
    @Override
    public boolean terminate(WorkerContext context, SCCAggrValue value)
        throws IOException {
        // If all vertices is inactive, job is over.
        if (! value.getActive()) {
            return true;

        // state machine
        switch (value.getStage()) {
        case STAGE_TRANSPOSE_1:
            value.setStage(STAGE_TRANSPOSE_2);
            break;
        case STAGE_TRANSPOSE_2:
```

```

        value.setStage(STAGE_TRIMMING);
        break;
    case STAGE_TRIMMING:
        value.setStage(STAGE_FW_START);
        break;
    case STAGE_FW_START:
        value.setStage(STAGE_FW_REST);
        break;
    case STAGE_FW_REST:
        if (value.getUpdated()) {
            value.setStage(STAGE_FW_REST);
        } else {
            value.setStage(STAGE_BW_START);

            break;
        }
    case STAGE_BW_START:
        value.setStage(STAGE_BW_REST);
        break;
    case STAGE_BW_REST:
        if (value.getUpdated()) {
            value.setStage(STAGE_BW_REST);
        } else {
            value.setStage(STAGE_TRIMMING);

            break;
        }

    value.setActive(false);
    value.setUpdated(false);
    return false;

```

```

public static class SCCVertexReader extends
GraphLoader<LongWritable, MyValue, NullWritable, LongWritable> {
    @ Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, MyValue, NullWritable, LongWritable> context)
        throws IOException {
        SCCVertex vertex = new SCCVertex();
        vertex.setId((LongWritable) record.get(0));

```

```

vertex.setID(new LongWritable(), record.get(0));
String[] edges = record.get(1).toString().split(",");
for (int i = 0; i < edges.length; i++) {
    try {
        long destID = Long.parseLong(edges[i]);
        vertex.addEdge(new LongWritable(destID), NullWritable.get());
    } catch (NumberFormatException nfe) {
        System.err.println("Ignore " + nfe);
    }
}

context.addVertexRequest(vertex);

public static void main(String[] args) throws IOException {
    if (args.length < 2) {
        System.out.println("Usage: <input> <output>");
        System.exit(-1);
    }

    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(SCCVertexReader.class);
    job.setVertexClass(SCCVertex.class);
    job.setAggregatorClass(SCCAggregator.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    long startTime = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
}

```

9.7.6. Connected component

If there is path between 2 vertices, it means the 2 vertices are connected. If any two vertices in undirected graph G are connected, G is called a connected graph. Otherwise, G is called an unconnected graph. A connected sub-graph with a large number of vertices is called a connected component.

This algorithm calculates connected component members of each vertex, and outputs the connected component of the vertex value that includes the smallest vertex ID. The smallest vertex ID is transmitted along edges to all vertices of the connected component.

Sample Code

Code for connecting components is as follows:

```
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.examples.SSP.MinLongCombiner;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.WritableRecord;

* Compute the connected component membership of each vertex and output
* each vertex which's value containing the smallest id in the connected
* component containing that vertex.

* Algorithm: propagate the smallest vertex id along the edges to all
* vertices of a connected component.

public class ConnectedComponents {
    public static class CCVertex extends
        Vertex<LongWritable, LongWritable, NullWritable, LongWritable> {
        @Override
        public void compute(
            ComputeContext<LongWritable, LongWritable, NullWritable, LongWritable> context,
            Iterable<LongWritable> msgs) throws IOException {
            if (context.getSuperstep() == 0L) {
                this.setValue(getId());
                context.sendMessageToNeighbors(this, getValue());
                return;

                long minID = Long.MAX_VALUE;
                for (LongWritable id : msgs) {
                    if (id.get() < minID) {
                        minID = id.get();
                    }
                }
            }
        }
    }
}
```

```

if (minID < this.getValue().get()) {
    this.setValue(new LongWritable(minID));
    context.sendMessageToNeighbors(this, getValue());
} else {
    this.voteToHalt();
}

```

* Output Table Description:

* | Field | Type | Comment |

* | v | bigint | vertex id |

* | minID | bigint | smallest id in the connected component |

@Override

```

public void cleanup(
    WorkerContext<LongWritable, LongWritable, NullWritable, LongWritable> context)
    throws IOException {
    context.write(getId(), getValue());
}

```

* Input Table Description:

* | Field | Type | Comment |

* | v | bigint | vertex id |

* | es | string | comma separated target vertex id of outgoing edges |

* Example:

* For graph:

* 1 ----- 2

* 3 ----- 4

* Input table:

* Input table:

```

    * | v | es |

    * | 1 | 2,3 |
    * | 2 | 1,4 |
    * | 3 | 1,4 |
    * | 4 | 2,3 |

public static class CCVertexReader extends
GraphLoader<LongWritable, LongWritable, NullWritable, LongWritable> {
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, NullWritable, LongWritable> context)
        throws IOException {
        CCVertex vertex = new CCVertex();
        vertex.setID((LongWritable) record.get(0));
        String[] edges = record.get(1).toString().split(",");
        for (int i = 0; i < edges.length; i++) {
            long destID = Long.parseLong(edges[i]);
            vertex.addEdge(new LongWritable(destID), NullWritable.get());

            context.addVertexRequest(vertex);
        }
    }

    public static void main(String[] args) throws IOException {
        if (args.length < 2) {
            System.out.println("Usage: <input> <output>");
            System.exit(-1);
        }

        GraphJob job = new GraphJob();
        job.setGraphLoaderClass(CCVertexReader.class);
        job.setVertexClass(CCVertex.class);
        job.setCombinerClass(MinLongCombiner.class);
        job.addInput(TableInfo.builder().tableName(args[0]).build());
        job.addOutput(TableInfo.builder().tableName(args[1]).build());
        long startTime = System.currentTimeMillis();
        job.run();
        System.out.println("Job Finished in "
            + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
    }
}

```

9.7.7. Topology Sorting

In directed edge (u,v) , all vertex sequences satisfying $u < v$ are called topological sequences. Topological sorting is an algorithm used to calculate the topological sequence of a directed graph.

Specifically, the algorithm:

1. Find a vertex that does not have any inbound edge from the graph and outputs the vertex.
2. Delete the vertex and all outbound edges from the graph.
3. Repeat the preceding steps until all vertices are output.

Sample Code

The code for the topology ordering algorithm is as follows:

```
import java.io.IOException;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.Combiner;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.graphjob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.BooleanWritable;
import com.aliyun.odps.io.WritableRecord;
public class TopologySort {
    private final static Log LOG = LogFactory.getLog(TopologySort.class);
    public static class TopologySortVertex extends
        Vertex<LongWritable, LongWritable, NullWritable, LongWritable> {
        @Override
        public void compute(
            ComputeContext<LongWritable, LongWritable, NullWritable, LongWritable> context,
            Iterable<LongWritable> messages) throws IOException {
            // in superstep 0, each vertex sends message whose value is 1 to its
```

```

// neighbors
if (context.getSuperstep() == 0) {
    if (hasEdges()) {
        context.sendMessageToNeighbors(this, new LongWritable(1L));

    } else if (context.getSuperstep() >= 1) {
        // compute each vertex's indegree
        long indegree = getValue().get();
        for (LongWritable msg : messages) {
            indegree += msg.get();

        }

        setValue(new LongWritable(indegree));
        if (indegree == 0) {
            voteToHalt();
            if (hasEdges()) {
                context.sendMessageToNeighbors(this, new LongWritable(-1L));

            }

            context.write(new LongWritable(context.getSuperstep()), getId());
            LOG.info("vertex: " + getId());

        }

        context.aggregate(new LongWritable(indegree));

    }

}

public static class TopologySortVertexReader extends
GraphLoader<LongWritable, LongWritable, NullWritable, LongWritable> {
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, NullWritable, LongWritable> context)
        throws IOException {
        TopologySortVertex vertex = new TopologySortVertex();
        vertex.setId((LongWritable) record.get(0));
        vertex.setValue(new LongWritable(0));
        String[] edges = record.get(1).toString().split(",");
        for (int i = 0; i < edges.length; i++) {
            long edge = Long.parseLong(edges[i]);
            if (edge >= 0) {
                vertex.addEdge(new LongWritable(Long.parseLong(edges[i])),
                    NullWritable.get());
            }
        }
    }
}

```

```
LOG.info(record.toString());
context.addVertexRequest(vertex);

public static class LongSumCombiner extends
Combiner<LongWritable, LongWritable> {
    @Override
    public void combine(LongWritable vertexId, LongWritable combinedMessage,
        LongWritable messageToCombine) throws IOException {
        combinedMessage.set(combinedMessage.get() + messageToCombine.get());

public static class TopologySortAggregator extends
Aggregator<BooleanWritable> {
    @SuppressWarnings("rawtypes")
    @Override
    public BooleanWritable createInitialValue(WorkerContext context)
        throws IOException {
        return new BooleanWritable(true);

    @Override
    public void aggregate(BooleanWritable value, Object item)
        throws IOException {
        boolean hasCycle = value.get();
        boolean inDegreeNotZero = ((LongWritable) item).get() == 0 ? false : true;
        value.set(hasCycle && inDegreeNotZero);

    @Override
    public void merge(BooleanWritable value, BooleanWritable partial)
        throws IOException {
        value.set(value.get() && partial.get());

    @SuppressWarnings("rawtypes")
    @Override
    public boolean terminate(WorkerContext context, BooleanWritable value)
        throws IOException {
        if (context.getSuperstep() == 0) {
            // since the initial aggregator value is true, and in superstep we don't
            // do aggregate
```

```

        // do aggregate
        return false;

        return value.get();

public static void main(String[] args) throws IOException {
    if (args.length != 2) {
        System.out.println("Usage : <inputTable> <outputTable>");
        System.exit(-1);

        // The input table is in the format of
        // 0 1, 2
        // 1 3
        // 2 3
        // 3 -1
        // The first column is vertexid, and the second column is the destination vertexid of the vertex edge
        // If the value is -1, the vertex does not have any outbound edge
        // The output table is in the format of
        // 0 0
        // 1 1
        // 1 2
        // 2 3
        // The first column is the supstep value, in which the topological sequence is hidden. The second column is vertexid
        // TopologySortAggregator is used to determine if the graph has loops
        // If the input graph has a loop, the iteration ends when the indegree of vertices in the active state is not 0
        // You can use records in the input and output tables to determine if the graph has loops
        GraphJob job = new GraphJob();
        job.setGraphLoaderClass(TopologySortVertexReader.class);
        job.setVertexClass(TopologySortVertex.class);
        job.addInput(TableInfo.builder().tableName(args[0]).build());
        job.addOutput(TableInfo.builder().tableName(args[1]).build());
        job.setCombinerClass(LongSumCombiner.class);
        job.setAggregatorClass(TopologySortAggregator.class);
        long startTime = System.currentTimeMillis();
        job.run();
        System.out.println("Job Finished in "
            + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
    }
}

```

9.7.8. Linear regression

In statistics, linear regression is a statistical analysis method used to determine the dependency between two or more variables. Linear regression is different from the classification algorithm that predicts discrete values

linear regression

The regression algorithm can predict continuous values. The linear regression algorithm defines the loss function as the sum of the least square errors of a sample set and solves the weight vector by minimizing the loss function.

A common solution is the gradient descent method, which is implemented in the following steps:

1. Initialize the weight vector to provide the descent speed and iterations or iteration convergence condition.
2. Calculate the least square error for each sample.
3. Calculate the sum of the least square errors and update the weight based on the descent speed.
4. Repeat iterations until convergence occurs.

Sample code

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;
/**
 * LineRegression input: y,x1,x2,x3,.....
 **/
public class LinearRegression {
```



```

public class LinearRegression {
    public static class GradientWritable implements Writable {
        Tuple lastTheta;
        Tuple currentTheta;
        Tuple tmpGradient;
        LongWritable count;
        DoubleWritable lost;
        @Override
        public void readFields(DataInput in) throws IOException {
            lastTheta = new Tuple();
            lastTheta.readFields(in);
            currentTheta = new Tuple();
            currentTheta.readFields(in);
            tmpGradient = new Tuple();
            tmpGradient.readFields(in);
            count = new LongWritable();
            count.readFields(in);
            /* update 1: add a variable to store lost at every iteration */
            lost = new DoubleWritable();
            lost.readFields(in);
        }
        @Override
        public void write(DataOutput out) throws IOException {
            lastTheta.write(out);
            currentTheta.write(out);
            tmpGradient.write(out);
            count.write(out);
            lost.write(out);
        }
    }
    public static class LinearRegressionVertex extends
        Vertex<LongWritable, Tuple, NullWritable, NullWritable> {
        @Override
        public void compute(
            ComputeContext<LongWritable, Tuple, NullWritable, NullWritable> context,
            Iterable<NullWritable> messages) throws IOException {
            context.aggregate(getValue());
        }
    }
    public static class LinearRegressionVertexReader extends
        GraphLoader<LongWritable, Tuple, NullWritable, NullWritable> {

```

```

@Override
public void load(LongWritable recordNum, WritableRecord record,
    MutationContext<LongWritable, Tuple, NullWritable, NullWritable> context)
    throws IOException {
    LinearRegressionVertex vertex = new LinearRegressionVertex();
    vertex.setId(recordNum);
    vertex.setValue(new Tuple(record.getAll()));
    context.addVertexRequest(vertex);
}
}

public static class LinearRegressionAggregator extends
    Aggregator<GradientWritable> {
    @SuppressWarnings("rawtypes")
    @Override
    public GradientWritable createInitialValue(WorkerContext context)
        throws IOException {
        if (context.getSuperstep() == 0) {
            /* set initial value, all 0 */
            GradientWritable grad = new GradientWritable();
            grad.lastTheta = new Tuple();
            grad.currentTheta = new Tuple();
            grad.tmpGradient = new Tuple();
            grad.count = new LongWritable(1);
            grad.loss = new DoubleWritable(0.0);
            int n = (int) Long.parseLong(context.getConfiguration()
                .get("Dimension"));
            for (int i = 0; i < n; i++) {
                grad.lastTheta.append(new DoubleWritable(0));
                grad.currentTheta.append(new DoubleWritable(0));
                grad.tmpGradient.append(new DoubleWritable(0));
            }
            return grad;
        } else {
            return (GradientWritable) context.getLastAggregatedValue(0);
        }
    }

    public static double vecMul(Tuple value, Tuple theta) {
        /* perform this partial computing: y(i)-hθ(x(i)) for each sample */
        /* value denote a piece of sample and value(0) is y */
        double sum = 0.0;
        for (int j = 1; j < value.size(); j++)
            sum += Double.parseDouble(value.get(j).toString())

```

```

        * Double.parseDouble(theta.get(j).toString());
    Double tmp = Double.parseDouble(theta.get(0).toString()) + sum
        - Double.parseDouble(value.get(0).toString());
    return tmp;
}
@Override
public void aggregate(GradientWritable gradient, Object value)
    throws IOException {
    /*
     * perform on each vertex--each sample i: set theta(j) for each sample i
     * for each dimension
     */
    double tmpVar = vecMul((Tuple) value, gradient.currentTheta);
    /*
     * update 2:local worker aggregate(), perform like merge() below. This
     * means the variable gradient denotes the previous aggregated value
     */
    gradient.tmpGradient.set(0, new DoubleWritable(
        ((DoubleWritable) gradient.tmpGradient.get(0)).get() + tmpVar));
    gradient.lost.set(Math.pow(tmpVar, 2));
    /*
     * calculate (y(i)-hθ(x(i))) x(i)(j) for each sample i for each
     * dimension j
     */
    for (int j = 1; j < gradient.tmpGradient.size(); j++)
        gradient.tmpGradient.set(j, new DoubleWritable(
            ((DoubleWritable) gradient.tmpGradient.get(j)).get() + tmpVar
            * Double.parseDouble(((Tuple) value).get(j).toString())));
}
@Override
public void merge(GradientWritable gradient, GradientWritable partial)
    throws IOException {
    /* perform SumAll on each dimension for all samples. */
    Tuple master = (Tuple) gradient.tmpGradient;
    Tuple part = (Tuple) partial.tmpGradient;
    for (int j = 0; j < gradient.tmpGradient.size(); j++) {
        DoubleWritable s = (DoubleWritable) master.get(j);
        s.set(s.get() + ((DoubleWritable) part.get(j)).get());
    }
    gradient.lost.set(gradient.lost.get() + partial.lost.get());
}
}

```

```

    }
    @SuppressWarnings("rawtypes")
    @Override
    public boolean terminate(WorkerContext context, GradientWritable gradient)
        throws IOException {
        /*
         * 1. calculate new theta 2. judge the diff between last step and this
         * step, if smaller than the threshold, stop iteration
         */
        gradient.set = new DoubleWritable(gradient.get()
            / (2 * context.getTotalNumVertices()));
        /*
         * we can calculate lost in order to make sure the algorithm is running on
         * the right direction (for debug)
         */
        System.out.println(gradient.count + " lost:" + gradient.get());
        Tuple tmpGradient = gradient.tmpGradient;
        System.out.println("tmpGra" + tmpGradient);
        Tuple lastTheta = gradient.lastTheta;
        Tuple tmpCurrentTheta = new Tuple(gradient.currentTheta.size());
        System.out.println(gradient.count + " terminate_start_last:" + lastTheta);
        double alpha = 0.07; // learning rate
        // alpha =
        // Double.parseDouble(context.getConfiguration().get("Alpha"));
        /* perform theta(j) = theta(j)-alpha*tmpGradient */
        long M = context.getTotalNumVertices();
        /*
         * update 3: add (/M) on the code. The original code forget this step
         */
        for (int j = 0; j < lastTheta.size(); j++) {
            tmpCurrentTheta
                .set(
                    j,
                    new DoubleWritable(Double.parseDouble(lastTheta.get(j)
                        .toString()
                        - alpha
                        / M
                        * Double.parseDouble(tmpGradient.get(j).toString()))));
        }
        System.out.println(gradient.count + " terminate_start_current:"
            + tmpCurrentTheta);
    }

```

```

// judge if convergence is happening.
double diff = 0.00d;
for (int j = 0; j < gradient.currentTheta.size(); j++)
    diff += Math.pow(((DoubleWritable) tmpCurrentTheta.get(j)).get()
        - ((DoubleWritable) lastTheta.get(j)).get(), 2);
if (/*
    * Math.sqrt(diff) < 0.00000000005d ||
    */Long.parseLong(context.getConfiguration().get("Max_Iter_Num")) == gradient.count
    .get()) {
    context.write(gradient.currentTheta.toArray());
    return true;
}
gradient.lastTheta = tmpCurrentTheta;
gradient.currentTheta = tmpCurrentTheta;
gradient.count.set(gradient.count.get() + 1);
int n = (int) Long.parseLong(context.getConfiguration().get("Dimension"));
/*
    * update 4: Important!!! Remember this step. Graph won't reset the
    * initial value for global variables at the beginning of each iteration
    */
for (int i = 0; i < n; i++) {
    gradient.tmpGradient.set(i, new DoubleWritable(0));
}
return false;
}
}

public static void main(String[] args) throws IOException {
    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(LinearRegressionVertexReader.class);
    job.setRuntimePartitioning(false);
    job.setNumWorkers(3);
    job.setVertexClass(LinearRegressionVertex.class);
    job.setAggregatorClass(LinearRegressionAggregator.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    job.setMaxIteration(Integer.parseInt(args[2])); // Numbers of Iteration
    job.setInt("Max_Iter_Num", Integer.parseInt(args[2]));
    job.setInt("Dimension", Integer.parseInt(args[3])); // Dimension
    job.setFloat("Alpha", Float.parseFloat(args[4])); // Learning rate
    long start = System.currentTimeMillis();
    job.run();
}

```

```
System.out.println("Job Finished in "
    + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
}
```

9.7.9. Triangle Count

This algorithm is used to calculate the number of triangles passing through each vertex.

The algorithm is implemented using the following steps:

1. Each vertex sends its ID to all outbound neighbors.
2. Store inbound and outbound neighbors and sends them to the outbound neighbors.
3. Calculate the number of endpoint intersections for each Edge, get the sum, and output the result to the table.
4. Get the sum of the output result in the table, divide it by 3, and get the number of triangles.

Sample code

Code for the triangle count algorithm are as follows:

```
import java.io.IOException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.Tuple;
import com.aliyun.odps.io.Writable;
import com.aliyun.odps.io.WritableRecord;

* Compute the number of triangles passing through each vertex.

* The algorithm can be computed in three supersteps:
* I. Each vertex sends a message with its ID to all its outgoing
* neighbors.
* II. The incoming neighbors and outgoing neighbors are stored and
* send to outgoing neighbors.
* III. For each edge compute the intersection of the sets at destination
```

* vertex and sum them, then output to table.

* The triangle count is the sum of output table and divide by three since

* each triangle is counted three times.

```
public class TriangleCount {
    public static class TCVertex extends
        Vertex<LongWritable, Tuple, NullWritable, Tuple> {
        @Override
        public void setup(
            WorkerContext<LongWritable, Tuple, NullWritable, Tuple> context)
            throws IOException {
            // collect the outgoing neighbors
            Tuple t = new Tuple();
            if (this.hasEdges()) {
                for (Edge<LongWritable, NullWritable> edge : this.getEdges()) {
                    t.append(edge.getDestVertexId());
                }

                this.setValue(t);
            }

            @Override
            public void compute(
                ComputeContext<LongWritable, Tuple, NullWritable, Tuple> context,
                Iterable<Tuple> msgs) throws IOException {
                if (context.getSuperstep() == 0L) {
                    // sends a message with its ID to all its outgoing neighbors
                    Tuple t = new Tuple();
                    t.append(getId());
                    context.sendMessageToNeighbors(this, t);
                } else if (context.getSuperstep() == 1L) {
                    // store the incoming neighbors
                    for (Tuple msg : msgs) {
                        for (Writable item : msg.getAll()) {
                            if (! this.getValue().getAll().contains((LongWritable)item)) {
                                this.getValue().append((LongWritable)item);
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
// send both incoming and outgoing neighbors to all outgoing neighbors
context.sendMessageToNeighbors(this, getValue());
} else if (context.getSuperstep() == 2L) {
    // count the sum of intersection at each edge
    long count = 0;
    for (Tuple msg : msgs) {
        for (Writable id : msg.getAll()) {
            if (getValue().getAll().contains(id)) {
                count ++;
            }
        }
    }

    // output to table
    context.write(getId(), new LongWritable(count));
    this.voteToHalt();
}

public static class TCVertexReader extends
GraphLoader<LongWritable, Tuple, NullWritable, Tuple> {
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, Tuple, NullWritable, Tuple> context)
        throws IOException {
        TCVertex vertex = new TCVertex();
        vertex.setID((LongWritable) record.get(0));
        String[] edges = record.get(1).toString().split(",");
        for (int i = 0; i < edges.length; i++) {
            try {
                long destID = Long.parseLong(edges[i]);
                vertex.addEdge(new LongWritable(destID), NullWritable.get());
            } catch (NumberFormatException nfe) {
                System.err.println("Ignore " + nfe);
            }
        }

        context.addVertexRequest(vertex);
    }

    public static void main(String[] args) throws IOException {
```



```

if (args.length < 2) {
    System.out.println("Usage: <input> <output>");
    System.exit(-1);

    GraphJob job = new GraphJob();
    job.setGraphLoaderClass(TCVertexReader.class);
    job.setVertexClass(TCVertex.class);
    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    long startTime = System.currentTimeMillis();
    job.run();
    System.out.println("Job Finished in "
        + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
}

```

9.7.10. Vertex Input

Sample code

```

import java.io.IOException;
import com.aliyun.odps.conf.Configuration;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.VertexResolver;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.VertexChanges;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.WritableComparable;
import com.aliyun.odps.io.WritableRecord;

```

* The following example describes how to compile a graph job program to load data of different types . It mainly describes how GraphLoader
* and VertexResolver are cooperated to build the graph.

* A MaxCompute Graph job uses MaxCompute tables as the input. Assume that a job has two tables a

s the input, one storing vertices and the other storing edges.

* The format of the table storing vertex information is as follows:

* | VertexID | VertexValue |

* | id0| 9|

* | id1| 7|

* | id2| 8|

* The format of the table storing edge information is as follows:

* | VertexID | DestVertexID| EdgeValue|

* | id0| id1| 1|

* | id0| id2| 2|

* | id2| id1| 3|

* The preceding two tables show that id0 has two outbound edges pointing to id1 and id2 respectively. id2 has an outbound edge pointing to id1, and id1 has no outbound edges.

* For data of this type, in `GraphLoader::load(LongWritable, Record, MutationContext)`,

* `MutationContext#addVertexRequest(Vertex)` can be used to add vertices to the graph, while

* `link MutationContext#addEdgeRequest(WritableComparable, Edge)` can be used to add edges to the graph. In

* `link VertexResolver#resolve(WritableComparable, Vertex, VertexChanges, boolean)`

* vertices and edges added in the `load()` method are combined to a vertex object, which is used as the return value and added to the graph for calculation.

```
public class VertexInputFormat {
    private final static String EDGE_TABLE = "edge.table";
```

* Resolve a record to vertices and edges. Each record indicates a vertex or an edge according to its source.

```

* Similar to com.aliyun.odps.mapreduce.Mapper#map,
* enter a record to generate key-value pairs. The keys are vertex IDs,
* and the values are vertices or edges written based on the context. These key-value pairs are summarized based on vertex IDs using LoadingVertexResolver.

* Note: Vertices or edges added here are requests sent based on the record content, and are not used in calculation. Only
* vertices or edges added using VertexResolver participate in calculation.

public static class VertexInputLoader extends
GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {
    private boolean isEdgeData;

    * Configure VertexInputLoader.

    * @param conf
    * Indicates the configuration parameters of a job, which are configured in the main GraphJob or set on the console.

    * @param workerId
    * Indicates the serial number of the operating Worker, which starts from 0 and can be used to build a unique vertex ID.

    * @param inputTableInfo
    * Indicates information about the input table load to the current Worker, which can be used to determine the type of currently input data, that is, the record format.

    @Override
    public void setup(Configuration conf, int workerId, TableInfo inputTableInfo) {
        isEdgeData = conf.get(EDGE_TABLE).equals(inputTableInfo.getTableName());

        * Based on the record content, resolve corresponding edges and send a request to add them to the graph.

        * @param recordNum
        * Indicates the record serial number, which starts from 1 and is separately counted in each Worker.

        * @param record
        * Indicates the record in the input table. It contains three columns, indicating the first vertex, last vertex, and edge weight.

        * @param context
        * Indicates the context, requesting to add resolved edges to the graph.

```

```
@Override
public void load(
    LongWritable recordNum,
    WritableRecord record,
    MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context)
    throws IOException {
    if (isEdgeData) {

        * Data is from the table that stores edge information.

        * 1. The first column indicates the first vertex ID.

        LongWritable sourceVertexID = (LongWritable) record.get(0);

        * 2. The second column indicates the last vertex ID.

        LongWritable destinationVertexID = (LongWritable) record.get(1);

        * 3. The third column indicates the edge weight.

        LongWritable edgeValue = (LongWritable) record.get(2);

        * 4. Create an edge that consists of the last vertex ID and edge weight.

        Edge<LongWritable, LongWritable> edge = new Edge<LongWritable, LongWritable>(
            destinationVertexID, edgeValue);

        * 5. Send a request to add an edge to the first vertex.

        context.addEdgeRequest(sourceVertexID, edge);

        * 6. If each record indicates a bidirectional edge, repeat steps 4 and 5. Edge<LongWritable, LongW
            ritable> edge2 = new
            * Edge<LongWritable, LongWritable>( sourceVertexID, edgeValue);
            * context.addEdgeRequest(destinationVertexID, edge2);

    } else {

        * Data comes from the table that stores vertex information.
```

* 1. The first column indicates the vertex ID.

```
LongWritable vertexID = (LongWritable) record.get(0);
```

* 2. The second column indicates the vertex value.

```
LongWritable vertexValue = (LongWritable) record.get(1);
```

* 3. Create a vertex that consists of the vertex ID and vertex value.

```
MyVertex vertex = new MyVertex();
```

* 4. Initialize the vertex.

```
vertex.setID(vertexID);
vertex.setValue(vertexValue);
```

* 5. Send a request to add a vertex.

```
context.addVertexRequest(vertex);
```

* Summarize key-value pairs generated using `GraphLoader::load(LongWritable, Record, MutationContext)`, which is similar to

* reduce in `com.aliyun.odps.mapreduce.Reducer`. For the unique vertex ID, all actions such as

* adding/deleting vertices or edges on the ID is stored in `VertexChanges`.

* Note: Not only conflicting vertices or edges added by using the `load()` method are called. (A conflict occurs when multiple same vertex objects or duplicate edges are added.)

* All IDs requested to be generated using the `load()` method are called.

```
public static class LoadingResolver extends
VertexResolver<LongWritable, LongWritable, LongWritable, LongWritable> {
```

* Process a request about adding/deleting vertices or edges for an ID.

* `VertexChanges` has four APIs, which correspond to the four APIs of `MutationContext`:

* `VertexChanges::getAddedVertexList()` corresponds to

* `MutationContext::addVertexRequest(Vertex)`.

* In the load() method, if vertex objects with the same ID are requested to be added, such vertex objects are collected to the return list.

* VertexChanges::getAddedEdgeList() corresponds to

* MutationContext::addEdgeRequest(WritableComparable, Edge)

* If edge objects with the same first vertex ID are requested to be added, such edge objects are collected to the return list.

* VertexChanges::getRemovedVertexCount() corresponds to

* MutationContext::removeVertexRequest(WritableComparable)

* If vertices with the same ID are requested to be deleted, the number of total deletion requests is returned.

* VertexChanges#getRemovedEdgeList() corresponds to

* MutationContext#removeEdgeRequest(WritableComparable, WritableComparable)

* If edge objects with the same first vertex ID are requested to be deleted, such edge objects are collected to the return list.

* By processing ID changes, you can state whether the ID participates in calculation using the return value. If the returned vertex is not null,

* the ID participates in subsequent calculation. If the returned vertex is null, the ID does not participate in subsequent calculation.

* @param vertexId

* Indicates the ID of the vertex requested to be added or first vertex ID of the edge requested to be added.

* @param vertex

* Indicates an existing vertex object. Its value is always null in the data loading phase.

* @param vertexChanges

* Indicates the set of vertices or edges requested to be added/deleted on the ID.

* @param hasMessages

* Indicates whether the ID has any input message. Its value is always false in the data loading phase.

@Override

```
public Vertex<LongWritable, LongWritable, LongWritable, LongWritable> resolve(
    LongWritable vertexId,
    Vertex<LongWritable, LongWritable, LongWritable, LongWritable> vertex,
    VertexChanges<LongWritable, LongWritable, LongWritable, LongWritable> vertexChanges,
    boolean hasMessages) throws IOException {
```

* 1. Obtain the vertex object for calculation.

MyVertex computeVertex = null;

```

MyVertex computeVertex = null;
if (vertexChanges.getAddedVertexList() == null
    || vertexChanges.getAddedVertexList().isEmpty()) {
    computeVertex = new MyVertex();
    computeVertex.setId(vertexId);
} else {

```

* Assume that each record indicates a unique vertex in the table storing vertex information.

```

computeVertex = (MyVertex) vertexChanges.getAddedVertexList().get(0);

```

* 2. Add the edge requested to be added to the vertex to the vertex object. If data is duplicated, perform deduplication based on the algorithm needs.

```

if (vertexChanges.getAddedEdgeList() != null) {
    for (Edge<LongWritable, LongWritable> edge : vertexChanges
        .getAddedEdgeList()) {
        computeVertex.addEdge(edge.getDestVertexId(), edge.getValue());
    }
}

```

* 3. Return the vertex object and add it to the final graph for calculation.

```

return computeVertex;

```

* Determine actions of the vertex that participates in calculation.

```

public static class MyVertex extends
    Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {

```

* Write the vertex edge to the result table according to the format of the input table. Ensure that the format and data of the input and output tables are the same.

```

* @param context
* Indicates the context during running.
* @param messages
* Indicates the input message.

```

```
@Override
public void compute(
    ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context,
    Iterable<LongWritable> messages) throws IOException {

    * Write the vertex ID and value to the result table storing vertices.

    context.write("vertex", getId(), getValue());

    * Write the vertex edge to the result table storing edges.

    if (hasEdges()) {
        for (Edge<LongWritable, LongWritable> edge : getEdges()) {
            context.write("edge", getId(), edge.getDestVertexId(),
                edge.getValue());
        }
    }

    * Perform one round of iteration.

    voteToHalt();

    * @param args
    * @throws IOException

    public static void main(String[] args) throws IOException {
        if (args.length < 4) {
            throw new IOException(
                "Usage: VertexInputFormat <vertex input> <edge input> <vertex output> <edge output>");
        }

        * GraphJob is used to configure Graph jobs.

        GraphJob job = new GraphJob();

        * 1. Specify input graph data and the table storing edge data.

        job.addInput(TableInfo.builder().tableName(args[0]).build());
```



```
job.addInput(TableInfo.builder().tableName(args[1]).build());
job.set(EDGE_TABLE, args[1]);
```

* 2. Specify the data loading mode, resolve the record as edges. Similar to the map, the generated key is the vertex ID, and the value is the edge.

```
job.setGraphLoaderClass(VertexInputLoader.class);
```

* 3. Specify the data loading phase, and generate the vertex for calculation. Similar to reduce, edges generated by map are combined to a vertex.

```
job.setLoadingVertexResolverClass>LoadingResolver.class);
```

* 4. Specify actions of the vertex that participates in calculation. The vertex.compute() method is used for each round of iteration.

```
job.setVertexClass(MyVertex.class);
```

* 5. Specify the output table of the Graph job, and write the calculation result to the result table.

```
job.addOutput(TableInfo.builder().tableName(args[2]).label("vertex").build());
job.addOutput(TableInfo.builder().tableName(args[3]).label("edge").build());
```

* 6. Submit the job for execution.

```
job.run();
```

9.7.11. Edge Input

Sample Code

```
import java.io.IOException;
import com.aliyun.odps.conf.Configuration;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.VertexResolver;
```

```
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.VertexChanges;
import com.aliyun.odps.graph.Edge;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.WritableComparable;
import com.aliyun.odps.io.WritableRecord;
```

* The following example describes how to compile a graph job program to load data of different types . It mainly describes how GraphLoader

* and VertexResolver are cooperated to build the graph.

* A MaxCompute Graph job uses MaxCompute tables as the input. Assume that a job has two tables as the input, one storing vertices and the other storing edges.

* The format of the table storing vertex information is as follows:

```
* | VertexID | VertexValue |
```

```
* | id0| 9|
```

```
* | id1| 7|
```

```
* | id2| 8|
```

* The format of the table storing edge information is as follows:

```
* | VertexID | DestVertexID| EdgeValue|
```

```
* | id0| id1| 1|
```

```
* | id0| id2| 2|
```

```
* | id2| id1| 3|
```

* The preceding two tables show that id0 has two outbound edges pointing to id1 and id2 respectively . id2 has an outbound edge pointing to id1, and id1 has no outbound edges.

* For data of this type, in GraphLoader::load(LongWritable, Record, MutationContext),

* MutationContext#addVertexRequest(Vertex) can be used to add vertices to the graph, while

* link MutationContext#addEdgeRequest(WritableComparable, Edge) can be used to add edges to the

graph. In

* link VertexResolver#resolve(WritableComparable, Vertex, VertexChanges, boolean)

* vertices and edges added in the load() method are combined to a vertex object, which is used as the return value and added to the graph for calculation.

```
public class VertexInputFormat {
```

```
    private final static String EDGE_TABLE = "edge.table";
```

* Resolve a record to vertices and edges. Each record indicates a vertex or an edge according to its source.

* Similar to com.aliyun.odps.mapreduce.Mapper#map,

* enter a record to generate key-value pairs. The keys are vertex IDs,

* and the values are vertices or edges written based on the context. These key-value pairs are summarized based on vertex IDs using LoadingVertexResolver.

* Note: Vertices or edges added here are requests sent based on the record content, and are not used for calculation. Only

* vertices or edges added using VertexResolver participate in calculation.

```
    public static class VertexInputLoader extends
```

```
        GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {
```

```
        private boolean isEdgeData;
```

* Configure VertexInputLoader.

* @param conf

* Indicates the configuration parameters of a job, which are configured in the main GraphJob or set on the console.

* @param workerId

* Indicates the serial number of the operating Worker, which starts from 0 and can be used to build a unique vertex ID.

* @param inputTableInfo

* Indicates information about the input table loaded to the current Worker, which can be used to determine the type of currently input data, that is, the record format.

@ Override

```
    public void setup(Configuration conf, int workerId, TableInfo inputTableInfo) {
```

```
        isEdgeData = conf.get(EDGE_TABLE).equals(inputTableInfo.getTableName());
```

* Based on the record content, resolve corresponding edges and send a request to add them to the graph.

* @param recordNum

* Indicates the record serial number, which starts from 1 and is separately counted in each Worker.

* @param record

* Indicates the record in the input table. It contains three columns, indicating the first vertex, last vertex, and edge weight.

* @param context

* Indicates the context, requesting to add resolved edges to the graph.

@Override

public void load(

 LongWritable recordNum,

 WritableRecord record,

 MutationContext<LongWritable, LongWritable, LongWritable, LongWritable> context)

 throws IOException {

 if (isEdgeData) {

 * Data comes from the table that stores edge information.

 * 1. The first column indicates the first vertex ID.

 LongWritable sourceVertexID = (LongWritable) record.get(0);

 * 2. The second column indicates the last vertex ID.

 LongWritable destinationVertexID = (LongWritable) record.get(1);

 * 3. The third column indicates the edge weight.

 LongWritable edgeValue = (LongWritable) record.get(2);

 * 4. Create an edge that consists of the last vertex ID and edge weight.

 Edge<LongWritable, LongWritable> edge = new Edge<LongWritable, LongWritable>(destinationVertexID, edgeValue);

 * 5. Send a request to add an edge to the first vertex.

```

context.addEdgeRequest(sourceVertexID, edge);

* 6. If each record indicates a bidirectional edge, repeat steps 4 and 5. Edge<LongWritable, LongW
ritable> edge2 = new
* Edge<LongWritable, LongWritable>( sourceVertexID, edgeValue);
* context.addEdgeRequest(destinationVertexID, edge2);

} else {

* Data comes from the table that stores vertex information.

* 1. The first column indicates the vertex ID.

LongWritable vertexID = (LongWritable) record.get(0);

* 2. The second column indicates the vertex value.

LongWritable vertexValue = (LongWritable) record.get(1);

* 3. Create a vertex that consists of the vertex ID and vertex value.

MyVertex vertex = new MyVertex();

* 4. Initialize the vertex.

vertex.setID(vertexID);
vertex.setValue(vertexValue);

* 5. Send a request to add a vertex.

context.addVertexRequest(vertex);

* Summarize key-value pairs generated using GraphLoader::load(LongWritable, Record, MutationCon
text), which is similar to
* reduce in com.aliyun.odps.mapreduce.Reducer. For the unique vertex ID, all actions such as
* adding/deleting vertices or edges on the ID is stored in VertexChanges.

* Note: Not only conflicting vertices or edges added by using the load() method are called. (A conflict

```

Note: Not only connecting vertices or edges added by using the load() method are called. (A conflict occurs when multiple same vertex objects or duplicate edges are added.)

- * All IDs requested to be generated using the load() method are called.

```
public static class LoadingResolver extends
```

```
VertexResolver<LongWritable, LongWritable, LongWritable, LongWritable> {
```

- * Process a request about adding/deleting vertices or edges for an ID.

- * VertexChanges has four APIs, which correspond to the four APIs of MutationContext:

- * VertexChanges::getAddedVertexList() corresponds to

- * MutationContext::addVertexRequest(Vertex).

- * In the load() method, if vertex objects with the same ID are requested to be added, such vertex objects are collected to the return list.

- * VertexChanges::getAddedEdgeList() corresponds to

- * MutationContext::addEdgeRequest(WritableComparable, Edge)

- * If edge objects with the same first vertex ID are requested to be added, such edge objects are collected to the return list.

- * VertexChanges::getRemovedVertexCount() corresponds to

- * MutationContext::removeVertexRequest(WritableComparable)

- * If vertices with the same ID are requested to be deleted, the number of total deletion requests is returned.

- * VertexChanges#getRemovedEdgeList() corresponds to

- * MutationContext#removeEdgeRequest(WritableComparable, WritableComparable)

- * If edge objects with the same first vertex ID are requested to be deleted, such edge objects are collected to the return list.

- * By processing ID changes, you can state whether the ID participates in calculation using the return value. If the returned vertex is not null,

- * the ID participates in subsequent calculation. If the returned vertex is null, the ID does not participate in subsequent calculation.

- * @param vertexId

- * Indicates the ID of the vertex requested to be added or first vertex ID of the edge requested to be added.

- * @param vertex

- * Indicates an existing vertex object. Its value is always null in the data loading phase.

- * @param vertexChanges

- * Indicates the set of vertices or edges requested to be added/deleted on the ID.

- * @param hasMessages

- * Indicates whether the ID has any input message. Its value is always false in the data loading phase.

e.

`@Override`

```
public Vertex<LongWritable, LongWritable, LongWritable, LongWritable> resolve(
    LongWritable vertexId,
    Vertex<LongWritable, LongWritable, LongWritable, LongWritable> vertex,
    VertexChanges<LongWritable, LongWritable, LongWritable, LongWritable> vertexChanges,
    boolean hasMessages) throws IOException {
```

* 1. Obtain the vertex object to participate in calculation.

```
MyVertex computeVertex = null;
if (vertexChanges.getAddedVertexList() == null
    || vertexChanges.getAddedVertexList().isEmpty()) {
    computeVertex = new MyVertex();
    computeVertex.setID(vertexId);
} else {
```

* Assume that each record indicates a unique vertex in the table storing vertex information.

```
computeVertex = (MyVertex) vertexChanges.getAddedVertexList().get(0);
```

* 2. Add the edge requested to be added to the vertex to the vertex object. If data may be duplicated, perform deduplication based on the algorithm needs.

```
if (vertexChanges.getAddedEdgeList() != null) {
    for (Edge<LongWritable, LongWritable> edge : vertexChanges
        .getAddedEdgeList()) {
        computeVertex.addEdge(edge.getDestVertexId(), edge.getValue());
    }
}
```

* 3. Return the vertex object and add it to the final graph for calculation.

```
return computeVertex;
```

* Determine actions of the vertex that participates in calculation.

```

public static class MyVertex extends
Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {

    * Write the vertex edge to the result table according to the format of the input table. Ensure that t
he format and data of the input and output tables are the same.

    * @param context
    * Indicates the context during running.
    * @param messages
    * Indicates the input message.

    @Override
    public void compute(
ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable> context,
        Iterable<LongWritable> messages) throws IOException {

        * Write the vertex ID and value to the result table storing vertices.

        context.write("vertex", getId(), getValue());

        * Write the vertex edge to the result table storing edges.

        if (hasEdges()) {
        for (Edge<LongWritable, LongWritable> edge : getEdges()) {
            context.write("edge", getId(), edge.getDestVertexId(),
                edge.getValue());

        * Perform one round of iteration.

        voteToHalt();

        * @param args
        * @throws IOException

        public static void main(String[] args) throws IOException {
            if (args.length < 4) {

```



```

if (args.length < 4) {
    throw new IOException(
        "Usage: VertexInputFormat <vertex input> <edge input> <vertex output> <edge output>");
}

* GraphJob is used to configure Graph jobs.

GraphJob job = new GraphJob();

* 1. Specify input graph data and the table storing edge data.

job.addInput(TableInfo.builder().tableName(args[0]).build());
job.addInput(TableInfo.builder().tableName(args[1]).build());
job.set(EDGE_TABLE, args[1]);

* 2. Specify the data loading mode, resolve the record as edges. Similar to the map, the generated key is the vertex ID, and the value is the edge.

job.setGraphLoaderClass(VertexInputLoader.class);

* 3. Specify the data loading phase, and generate the vertex that participates in calculation. Similar to reduce, edges generated by map are combined to a vertex.

job.setLoadingVertexResolverClass>LoadingResolver.class);

* 4. Specify actions of the vertex that participates in calculation. The vertex.compute() method is used for each round of iteration.

job.setVertexClass(MyVertex.class);

* 5. Specify the output table of the Graph job, and write the calculation result to the result table.

job.addOutput(TableInfo.builder().tableName(args[2]).label("vertex").build());
job.addOutput(TableInfo.builder().tableName(args[3]).label("edge").build());

* 6. Submit the job for execution.

job.run();

```

10.View Job Running Information

10.1. Logview

Logview is a tool to view and debug tasks once the MaxCompute job is submitted.

Using Logview, you can see the following details about a job:

- The Run Status of the task.
- The operation result of the task.
- Details of the task and the progress of each step.

When the job is submitted to MaxCompute, a link to the Logview is generated. You can open the Logview link directly on the browser to view information about the job for each job. The Logview page is valid for seven days.

Features

The following is a combination of a specific Logview web UI interface to introduce you to each component.

□

A Logview home page is divided into two upper and lower sections:

- Instance information
- Task information

Instance info

On the Logview home page, the upper half is the MaxCompute instance that you submit to generate SQL. A unique ID is generated after each SQL commit. Latency refers to the amount of time it takes to run, and the latency of other pages is similar.

The following are the four states:

- **Waiting:** Indicates that the current job is being processed in MaxCompute and is not committed to Fuxi to run.
- **Waiting List:** N indicates that the job was submitted to Fuxi and queued in Fuxi, is in the n-bit in the queue.
- **Running:** The Job runs in Fuxi.
- **Terminated:** The job has ended with no queue information.

Click the non-terminated status of a job to view detailed queue information.

Click status to view queue details:

- **Sub status:** The current sub-status information.
- **Waitpos:** The queuing location, where 0 indicates that it is running, and (-) indicates that it has not yet arrived Fuxi.
- **Queuing length:** The total queue length in the Fuxi.
- **Total priority:** The priority granted by the job runtime after it has been judged by the system.
- **SubStatus history:** When clicked, you can view the detailed history of job execution. It contains status codes, status descriptions, start time, duration, and so on. (Currently, some versions have no historical information.)

□

Task information

In the Logview home page, the lower section is the task description followed by the result description and other details.

Result:

You can view the result after a job executed, such as the results of a select SQL as shown in the following figure:

□

Detail:

After a job is executed, click detail to view the running status of the task.

□

- A MaxCompute task consists of one or more Fuxi jobs. For example, when your SQL task is complex, MaxCompute goes to Fuxi and submit multiple Fuxi jobs.
- Each Fuxi job consists of one or more Fuxi tasks. Simple Map Reduce usually produces two Fuxi tasks, namely Map and Reduce. You can view the two Fuxi task names as M1 and R2, respectively. When SQL is complex, more than one Fuxi may generate Task as shown in the preceding figure.
- Each task displays name of the task. For example, M1 is a map task, 4 In r54 means that it relies on J4. Execution will not begin until the last execution is complete. Similarly, j4_1_2_3 indicates that join4 has to rely on M1, M2, M3 three tasks to start the operation completely.
- I/O Records represents the number of records for the input and output of this task.

Click any Fuxi task to view the Fuxi instance content, as shown in the following figure:

□

Each Fuxi task consists of one or more Fuxi instances. When your input data levels are large MaxCompute activates more nodes to process the data. Each node is a Fuxi instance. Double-click the right-side column of the Fuxi task to view, or double-click the row to open the specific Fuxi instance information.

Towards the lower-end of the page, Logview is grouped for different stages of instance. Select the failed column to view the wrong node.

In the stdout and stderr columns, you can view standard output and standard error messages along with the information to be printed.

Troubleshooting through Logview

• Wrong tasks

When a task error occurs, a prompt message for the errors in the result on the Logview page pops up. Use the detail page Stderr for Fuxi instance to view information about a specific instance error.

• Data skew

Slow operation is usually because of individual instances in all Fuxi instances of a certain Fuxi task. Long Tail is caused by the uneven allocation of tasks within the same task. You can view the run results in the summary tab after the task runs. The output of each task is as follows:

output records:

R2_1_Stg1: 199998999 (min: 22552459, max: 177446540, avg: 99999499)

In the preceding figure, the large difference between min and max suggests that a data tilt has occurred at this stage. Meaning, if one word with high frequency appears, a tilt appears when you join this word.

10.2. Use Logview V2.0 to view job running information

This topic describes the entry and features of Logview V2.0. You can use Logview V2.0 to view job running information.

Overview

Logview V2.0 provides a newly designed UI, increases the loading speed, and delivers the following new features:

- Provides an interactive directed acyclic graph (DAG) to display the logical architecture of job processing. You can also view the operators of a job.
- Supports job execution playback.
- Allows you to view memory usage and CPU utilization by using Fuxi Sensor.

Entry

After you use the MaxCompute client (odpscmd) to submit a job, the system generates a Logview URL. Enter the Logview URL in the search engine and press Enter. On the Logview UI, click [Go to Logview V2.0](#).



Logview V2.0 page



No.	Section
1	The title and function section. For more information, see Title and function section .
2	The Basic Info section. For more information, see Basic Info .
3	The job details section. For more information, see Job details .

Title and function section

This section shows the job ID and job name. The job ID uniquely identifies a MaxCompute job. It is generated when you submit the job. The job name is displayed only if the job is submitted by using SDKs. You can also click the icons on the right of this section to perform operations.

Icon	Description
	Open the Logview_detail.txt file that contains job details. The file is saved on your computer.

Icon	Description
	Return to the original Logview UI.
	Save the job details as a file.

Basic Info

This section shows the basic information about a job.

Parameter	Description
MaxCompute Service	The endpoint of MaxCompute on which the job is running. For more information, see Configure endpoints .
Project	The name of the MaxCompute project to which the job belongs.
Cloud account	The Alibaba Cloud account that is used to submit the job.
Type	The type of the job. Valid values: SQL, SQLRT, LOT, XLib, CUPID, AlgoTask, and Graph.
Status	The status of the job. Valid values: • Success: The job is successful. • Failed: The job fails. • Canceled: The job is canceled. • Waiting: The job is being processed in MaxCompute but is not submitted to Job Scheduler. • Running: The job is being processed in Job Scheduler. • Terminated: The job is completed.
Start Time	The time when the job is submitted.
End Time	The time when the job is completed.
Latency	The period during which the job is executed.
Progress	The progress of the job.
Priority	The priority of the job.
Queue	The position of the job in the queue in the resource quota group.

Job details

In the job details section, you can query details about a job. This section consists of the following tabs:

- Job Details
 - Progress chart

In the upper part of the **Job Details** tab, the progress chart of a job is displayed. The progress chart shows the subtask dependencies from three dimensions: Fuxi jobs, Fuxi tasks, and operators. It also provides a series of tools to help you troubleshoot issues. The following figure shows the upper part of the job Details tab.



No.	Description
1	The breadcrumb navigation that is used to switch Fuxi jobs. JOB: _SQL_0_0_0_job_0 is the name of a Fuxi job.
2	The troubleshooting tool. You can use Progress Chart, Input Heat Chart, Output Heat Chart, TaskTime Heart Chart, or InstanceTime Heart Chart for troubleshooting.
3	You can click the  icon to refresh the job status and the  icon to zoom in or out the progress chart. You can also click the  icon to obtain MaxCompute Studio documentation and the  icon to switch to the upper level of the job.
4	The zoom tool.
5	The Fuxi task. A MaxCompute job consists of one or more Fuxi jobs. Each Fuxi job consists of one or more Fuxi tasks. Each Fuxi task consists of one or more Fuxi instances. If the amount of input data increases, MaxCompute starts more nodes for each task to process the data. A node is equivalent to a Fuxi instance. For example, a simple MapReduce job generates two Fuxi tasks: map task (M1) and reduce task (R2). If an SQL statement is complex, multiple Fuxi tasks may be generated. You can view the name of each Fuxi task on the execution page. For example, M1 indicates a map task. The 3 and 9 fields in R4_3_9 indicate that the map task can be executed only after M3 and C9_3 are completed. Similarly, M2_4_9_10_16 indicates that the M2 task can be executed only after R4_3_9, C9_3, R10_1_16, and C16_1 are completed. R/W indicates the numbers of rows that the task reads and writes. Click or right-click a task to view operator dependencies and operator graphs of the task. Logview V2.0 provides table dependencies so that you can quickly view the input and output tables.
6	The Fuxi task playback. You can click the  icon to start or stop the playback. You can drag the progress bar to play at certain time. The start time and end time are displayed on both sides of the progress bar. The playing time is displayed in the middle.
7	EagleEye.

Note

- The playback feature is not applicable to in-progress Fuxi tasks.
- An AlgoTask job, such as a Machine Learning Platform for AI (PAI) job, contains only one Fuxi task. Therefore, no progress charts are provided for such jobs.
- For non-SQL jobs, only Fuxi jobs and Fuxi tasks are displayed.
- If only one Fuxi job exists, the progress chart shows the dependencies among Fuxi tasks. If multiple Fuxi jobs exist, the progress chart shows the dependencies among the Fuxi jobs.

Running status of jobs

In the lower part of the **Job Details** tab, the detailed running information about the job is displayed. The following figure shows the lower part of the Job Details tab.



No.	Description
1	The Fuxi Jobs tab. You can switch Fuxi jobs on this tab.
2	The details about Fuxi tasks of the Fuxi job. Click a Fuxi task to display information about the Fuxi instance of this task. By default, information about the Fuxi instance of the first Fuxi task for the first Fuxi job is displayed.
3	Logview divides instances into groups based on their status. You can click the number next to Failed to query information about faulty nodes.
4	Fuxi Sensor. This feature is provided only for AlgoTask jobs, such as PAI jobs. You can use Fuxi Sensor to view the memory usage and CPU utilization of Fuxi instances. For more information, see Fuxi Sensor . Note Fuxi Sensor is available in the China (Chengdu), China (Shenzhen), China (Shanghai), China (Hangzhou), China (Zhangjiakou-Beijing Winter Olympics), and China (Beijing) regions.
5	StdOut and StdErr. You can view output messages, error messages, and information to be displayed. You can also download the information.
6	Debug. You can debug and troubleshoot errors.

Result

This tab shows the result of a job. If a job failed, this tab is displayed and shows the cause of the failure. The system displays data in plain text or a table based on the response format configured in MaxCompute. You can log on to the MaxCompute client and set the `odps.sql.select.output.format` parameter to specify the format. If you set this parameter to `HumanReadable`, the result is displayed in plain text. Otherwise, the result is displayed in CSV format.

- Table



You can perform the following operations on this tab:

- Select a row to export its data.
- Select the table heading to export data from the current page.
- In the table heading, click the  icon and select **Select All Data** or **Deselect All** as needed.
- Click export to export data in CSV format. CSV is a file name extension. You can use a text file or Sublime Text to open CSV files.

- Plain text



- SourceXML

This tab shows the source XML information about the job that is submitted to MaxCompute.

- SQL Script

This tab shows SQL scripts of the job that is submitted to MaxCompute.

- Summary

This tab shows the overall running information about the job that is submitted to MaxCompute.

- Json Summary

This tab shows the overall running information about the job in JSON format.

- History

This tab shows the historical information about a Fuxi instance if the instance is executed multiple times.

- Substatus History

This tab shows the detailed information about a job, including the status code, status description, start time, duration, and end time.

Fuxi Sensor

Fuxi Sensor is a resource view that shows a MaxCompute job from all dimensions. You can use Fuxi Sensor to view the memory usage and CPU utilization of a Fuxi instance. You can also use Fuxi Sensor to locate job issues and analyze running performance. For example, you can use Fuxi Sensor in the following scenarios:

- If out-of-memory (OOM) occurs, analyze the amount of memory used.
- Compare the numbers of requested and used resources to optimize the resource request process.

For example, you can use Fuxi Sensor to view the resource usage of a Fuxi instance.

- CPU utilization

The `cpu_usage` chart has two lines. One indicates the number of requested CPUs (`cpu_plan`), and the other one indicates the number of used CPUs (`cpu_usage`). In the y-axis, 400 indicates four processors.



- **Memory usage**

The `mem_usage` chart has two lines. One indicates the number of requested memory resources (`mem_plan`), and the other one indicates the number of used memory resources (`mem_usage`).

`mem_usage` contains Resident Set Size (RSS) and PageCache. If the memory is insufficient, RSS cannot be reclaimed, which may cause OOM. PageCache is the memory occupied by the kernel to cache files that are required by the read and write requests, such as log files. If the memory is insufficient, PageCache can be reclaimed.

- **Memory details**



- **RSS usage**



- **PageCache usage**



10.3. Errors and warnings using the MaxCompute compiler

The MaxCompute compiler is based on the next-generation SQL engine called MaxCompute2.0, which dramatically enhances SQL. It makes the process of language compilation and the ability of language expression easier. This article introduces you to the enhanced uses of the compiler.

Compiler ease of use improvements

To fully demonstrate the ease-of-use improvements of the MaxCompute compiler, it is recommended that you use MaxCompute studio together.

First, install MaxCompute Studio by adding a MaxCompute project and creating a project, and then creating a new MaxCompute. The script is as follows:

□

The following issues are detected in the preceding figure:

- An error with the `wm_concat` function can be seen in the First insert statement.
- When MaxCompute compares bigint and double data, it converts all data to double. This conversion from string to double, may cause error when SQL is executed. However, MaxCompute warns you whether you want to trigger this operation.

Point the mouse cursor on an error or warning prompts directly, for a specific error or warning message. If you do not modify the error and commit directly, it is blocked by MaxCompute studio, as shown in the following figure:

□

So, follow the prompts to modify the errors and warnings as follows:

□

After the modification, submit the script again, and you can now run it smoothly.

You can also use MaxCompute studio to set all warnings as errors, as shown in the following figure:

□

With the preceding settings, it is guaranteed that you won't accidentally miss out on any possible errors.

It is recommended that you use MaxCompute studio before submitting any scripts. The script is checked for static compilation, and we strongly recommend that you set the warning as an error. Modify all warnings before you submit the script to save time and resources. In addition, when an error script is submitted, it is pushed to your calculation health score. This reduces the priority of the future tasks. Moreover, future unmodified warnings also get incorporated into the health system. Meaning, the use of MaxCompute compiler and studio can never be degraded.

In many scenarios, you may receive warnings stating that an implicit type conversion is unsafe. However, if you need this conversion, eliminate the warnings by cast (xxx As); Use MaxCompute or a compiler to resolve this problem.

11. Open-source features of MaxCompute

MaxCompute is a fast and fully managed one-stop data warehouse solution developed by Alibaba. MaxCompute can process terabytes or petabytes of data. This topic describes open-source features of MaxCompute.

SDK

MaxCompute provides Java SDK and Python SDK interfaces to create, view, and delete MaxCompute tables. You can use the SDKs to manage MaxCompute by editing code. MaxCompute provides the following SDKs:

- Java SDK

For more information about how to use the Java SDK, see [SDK Reference > Java SDK](#).

For more information about commonly used core interfaces of MaxCompute, see [Java SDK](#). For more information, see [ODPS SDK for Java Developers](#) on GitHub.

How to obtain service support: Visit the official documentation or [submit a ticket](#) online.

- Python SDK

PyODPS is the Python SDK of MaxCompute. PyODPS supports the DataFrame framework and basic operations on MaxCompute objects. You can use PyODPS to analyze data in MaxCompute. For more information, see [aliyun-odps-python-sdk](#) on GitHub and [PyODPS documentation](#), which describes all related interfaces and classes in detail.

- For more information about how to use PyODPS to develop code, see the [PyODPS developer guide](#). Install PyODPS before you can use it. For more information, see the [PyODPS installation guide](#).
- For more information about the PyODPS community, see [PyODPS-related articles](#).
- You are welcome to help build the PyODPS ecosystem. For more information, see [PyODPS DataFrame overview](#).
- Feel free to share your feedback and suggestions to help PyODPS evolve. For more information, see [ODPS Python SDK and data analysis framework](#) on GitHub.

How to obtain service support: Visit the official documentation or [submit a ticket](#) online.

MaxCompute RODPS

RODPS is a plug-in that MaxCompute provides for R. For more information, see [ODPS Plugin for R](#) on GitHub.

How to obtain service support: Leave a message or create an issue in [ODPS Plugin for R](#) on GitHub.

MaxCompute JDBC

MaxCompute JDBC is an official Java Database Connectivity (JDBC) driver provided by MaxCompute. MaxCompute JDBC provides a set of interfaces for Java programs to execute SQL tasks. The project is hosted in [ODPS JDBC](#) on GitHub.

How to obtain service support: Leave a message or create an issue in [ODPS JDBC](#) on GitHub.

Data collector

MaxCompute provides a set of open-source data collectors.

MaxCompute provides data collectors for the following services:

- Flume
- Oracle GoldenGate (OGG)
- Sqoop
- Kettle
- Hive Data Transfer UDTF

The Flume and OGG data collectors are implemented based on the DataHub SDK, whereas the data collectors for Sqoop, Kettle, and Hive Data Transfer UDTF are implemented based on the Tunnel SDK. DataHub is a real-time data transfer channel, and Tunnel is a batch data transfer channel. The Flume and OGG data collectors are used to transfer data in real time. The data collectors for Sqoop, Kettle, and Hive Data Transfer UDTF are used to transfer data in batches in offline mode.

For more information about the source code, see [Aliyun MaxCompute Data Collectors](#) on GitHub. For more information about data collectors, see [wiki](#).

How to obtain service support: Leave a message or create an issue in [Aliyun MaxCompute Data Collectors](#) on GitHub.