Alibaba Cloud MaxCompute

User Guide

Issue: 20190122

MORE THAN JUST CLOUD | Alibaba Cloud

Legal disclaimer

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

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed due to product version upgrades, adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall 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 the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. 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 bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, 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 content of the Alibaba Cloud website, including but not limited to works, products, images, archives, information, materials, website architecture, website graphic layout, and webpage design, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectu

al property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of the Alibaba Cloud website, product programs, or content 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 contact Alibaba Cloud directly if you discover any errors in this document.

Generic conventions

Table -1: Style conventions

Style	Description	Example
	This warning information 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.
	This warning information 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 restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructio ns, best practices, tips, and other content that is good to know for the user.	**Note:** You can use **Ctrl** + **A** to select all files.
>	Multi-level menu cascade.	**Settings** > **Network** > **Set network type**
Bold	It is used for buttons, menus, page names, and other UI elements.	Click **OK**.
`Courier font`	It is used for commands.	Run the `cd /d C:/windows` command to enter the Windows system folder.
`Italics`	It is used for parameters and variables.	`bae log list --instanceid` *`Instance_ID`*
[] or [a|b]	It indicates that it is a optional value, and only one item can be selected.	`ipconfig` *`[-all|-t]`*
{} or {a|b}	It indicates that it is a required value, and only one item can be selected.	`swich` *`{stand | slave}`*

Contents

Legal disclaimer I
Generic conventions I
1 Definition 1
1.1 The glossary of Maxcompute 1
1.2 Project 4
1.3 Table 5
1.4 Partition 6
1.5 Data types 9
1.6 Lifecycle 13
1.7 Resource 14
1.8 Function 15
1.9 Task 15
1.10 Instance 16
2 Common commands 17
2.1 List of common commands 17
2.2 project operations 18
2.3 Table operations 21
2.4 Instances 25
2.5 Resources 30
2.6 Functions 32
2.7 Set operation 34
2.8 Other operations 35
3 Data upload and download 37
3.1 Data upload and download 37
3.2 Connection to data tunnel service 38
3.3 Cloud data migration 38
3.4 Data upload and download tools 39
3.5 Tunnel commands 41
3.6 Tunnel SDK 51
3.6.1 Summary 51
3.6.2 TableTunnel 52
3.6.3 UploadSession 54
3.6.4 DownloadSession 56
3.6.5 TunnelBufferedWriter 57
3.7 Bulk data channel SDK example 58
3.7.1 Example 58
3.7.2 简单下载示例 58
3.7.3 Example for multi-thread uploading 60
3.7.4 Example for multi-thread downloading 63
3.7.5 Example for BufferedWriter uploading 66

3.7.6 Example for BufferedWriter multi-thread uploading....66
3.8 Import or export data using the Data Integration....67
3.9 Real-time data tunnel of DataHub....71
4 SQL....72
4.1 SQL summary....72
4.2 Operators....73
4.3 Type conversions....77
4.4 DDL SQL....83
4.4.1 Table Operations....83
4.4.2 Lifecycle of table....89
4.4.3 Column and Partition operation....90
4.4.4 View operations....94
4.5 Insert Operation....95
4.5.1 INSERT OVERWRITE/INTO....95
4.5.2 MULTI INSERT....97
4.5.3 DYNAMIC PARTITION....98
4.5.4 VALUES....100
4.6 Lateral View....103
4.7 Select Operation....106
4.7.1 Introduction to the SELECT Syntax....106
4.7.2 SELECT Sequence....110
4.7.3 Subquery....111
4.7.4 UNION ALL/UNION [DISTINCT]....113
4.7.5 JOIN operation....114
4.7.6 SEMI JOIN....116
4.7.7 MAPJOIN HINT....116
4.7.8 HAVING clause....118
4.7.9 Explain....118
4.7.10 Common table expression (CTE)....121
4.8 Builtin functions....122
4.8.1 Date functions....122
4.8.2 Mathematical functions....139
4.8.3 Window functions....160
4.8.4 Aggregate functions....176
4.8.5 String functions....183
4.8.6 Other functions....205
4.9 UDF....226
4.9.1 UDF Summary....226
4.9.2 Java UDF....228
4.9.3 Python UDF....239
4.10 UDT....246
4.11 UDJ....256
4.12 Differences with other SQL syntax....267
4.13 SQL limits....268
4.14 Appendix....270

4.14.1 Escape characters.....270
4.14.2 LIKE usage.....271
4.14.3 Regular expression.....272
4.14.4 Reserved words and keywords.....274
4.14.5 Hive data type mapping table.....275
5 MapReduce.....276
5.1 Summary.....276
5.1.1 MapReduce.....276
5.1.2 Extended MapReduce.....279
5.1.3 Open-source MapReduce.....280
5.2 Function Introduction.....284
5.2.1 Basic concepts.....285
5.2.2 Commands.....285
5.2.3 Input and Output.....287
5.2.4 Resources.....288
5.2.5 Local run.....288
5.3 Program Example.....291
5.3.1 WordCount samples.....291
5.3.2 MapOnly samples.....294
5.3.3 Multi-input and Output.....296
5.3.4 Multi-task samples.....300
5.3.5 Secondary Sort samples.....303
5.3.6 Resource samples.....305
5.3.7 Counter samples.....308
5.3.8 Grep samples.....310
5.3.9 Join samples.....314
5.3.10 Sleep samples.....317
5.3.11 Unique samples.....318
5.3.12 Sort samples.....321
5.3.13 Partition samples.....324
5.3.14 Pipeline samples.....325
5.4 Java SDK.....328
5.4.1 Java SDK.....328
5.4.2 Overview of compatible versions of the SDK.....334
5.5 MR limits.....353
6 Java Sandbox.....358
7 External table.....362
7.1 Overview of External tables.....362
7.2 OSS STS mode authorization.....363
7.3 Access OSS unstructured data.....365
7.4 Processing open source format data for OSS.....377
7.5 Unstructured data exported to OSS.....384
7.6 Access Table Store data.....392
8 Security.....400

8.1 Target users....400
8.2 Quick Start....400
8.2.1 Use case: Add users and grant permissions....400
8.2.2 Use case: Add users and grant permissions using ACL....400
8.2.3 Use case: Project data protection....401
8.3 User authentication....402
8.4 User management....403
8.5 Role management....407
8.6 Authorization....410
8.7 Permission check....414
8.8 Security configurations....415
8.9 Data protection of projects....416
8.10 Security command list....419
8.10.1 Security configuration of a project....419
8.10.2 Manage permissions....420
8.10.3 Package-based resource sharing....421
8.11 Resource share across project space....422
8.11.1 Resource sharing across projects based on package....422
8.11.2 Package usage method....423
8.12 Column-level access control....426
9 Graph....432
9.1 Summary....432
9.2 Aggregator....435
9.3 Function overview....443
9.4 SDK summary....446
9.5 Development and debugging....447
9.6 Limits....455
9.7 Examples....456
9.7.1 SSSP....456
9.7.2 PageRank....460
9.7.3 Kmeans....462
9.7.4 BiPartiteMatchiing....468
9.7.5 Strongly-connected component....471
9.7.6 Connected component....478
9.7.7 Topology Sorting....481
9.7.8 Linear Regression....484
9.7.9 Triangle Count....488
9.7.10 Vertex Input....490
9.7.11 Edge Input....497
10 Lightning....504
10.1 Lightning overview....504
10.2 Activate Lightning service....506
10.3 Quick Start....507
10.3.1 Guide description....507

10.3.2 Prerequisites..... 507
10.3.3 Prepare client tools for connection..... 507
10.3.4 Access services and perform analysis..... 508
10.4 Access domain name..... 509
10.5 Access services using JDBC interfaces..... 510
10.5.1 Configure JDBC connections..... 510
10.5.2 Access services using common tools..... 511
10.5.3 JDBC driver..... 516
10.6 SQL reference..... 517
10.7 View tasks..... 519
10.8 Constraints and limitations..... 520
10.9 Lightning FAQs..... 520
11 MaxCompute Manager..... 523

1 Definition

1.1 The glossary of Maxcompute

This article lists the common concepts and terminologies of MaxCompute. For detailed description, please refer to the link in this article.

A

- AccessKey

  Access Key (AK for short, including Access Key Id and Access Key Secret) is the key to access the Aliyun API. After registering cloud account on Ali cloud official website,it can be generated on the Accesskeys management page to identify users and do signature verification for accessing MaxComputer or other cloud products. Access Key Secret must be kept secret.

- Security

  MaxCompute multi-tenant data security system mainly includes user authentication, user and authorization management in project space, resource sharing across project space and data protection in project space. For more details on MaxCompute security operation, see *Safety Guide*.

C

- Console

  MaxCompute Console is a client tool running under Windows/Linux. It can submit commands to complete project management, DDL, DML and other operations through Console. For tool installation and common parameters, See the *Client* for tool installations and common parameters.

D

- Data Type

  The data type corresponding to all columns in the MaxCompute table. For data types currently supported, see *Basic Concepts>Data types*.

- DDL

  Data Definition Language.Like creating tables, creating views, and so on, MaxCompute Div syntax see *User's Guide>DDL*.

- DML

  Data Manipulation Language. For example, INSERT operations, MaxCompute DML syntax, please see *Insert Operation* .

F

- Fuxi

  Fuxi is the module responsible for resource management and task scheduling in the core of Flying Platform. It also provides a basic programming framework for application development. The bottom task scheduling module of MaxCompute is the scheduling module of Fuxi.

I

- Instance (Instance)

  A specific instance of a job that represents a job that actually runs, similar to the concept of a job in hadoop. See *Basic Concepts>Task Instance* for details.

M

- MapReduce

  MaxCompute a programming model for processing data, usually used for parallel operation of large data sets. You can use the interface provided by MapReduce (Java API) to write MapReduce programs to process data in MaxCompute. The idea of programming is to divide data processing methods into Map (mapping) and Reduce (Protocol).

  Before formally executing Map, a partition is required. A slice is a cut of input data into equal-size blocks, each of which acts as a single map. The input of the worker is processed so that multiple Map Worker can work together.Each Map Worker reads in its own data, calculates and processes them, and finally integrates the intermediate results through the Reduce function to get the final results. For details, refer to the *User's guide>MapReduce*.

O

- ODPS

  ODPS is the original name of MaxCompute.

P

- Partition (partition)

  Partition partition refers to a table, based on the partition field (one or more combinations) divide the data store. That is, if the table does not have a partition, the data is placed directly

under the directory where the table is located. If a table has a partition, each partition corresponds to one of the directories in the table, the data is stored separately in a different partition directory. For more information about partitions, see *Basic Concepts>Partitions*.

- Project (Project)

  Project is the basic organizational unit of MaxCompute. It is similar to the concept of database or Scheme in traditional database, and is the main boundary of multi-user isolation and access control. For details, please see *Basic Concepts>Project* .

R

- Role (role)

  Roles are concepts used in MaxCompute security functions and can be seen as a collection of users with the same privileges. Multiple users can appear at one role at the same time, and a user can also belong to multiple roles. After all roles are authorized, all users under the role have the same permissions. For more information about role management, please see *User's guide>Role management* .

- Resource (resources)

  Resource (Resource) is a unique concept in MaxCompute. If you want to use MaxCompute's custom function (UDF) or MapReduce function, you need to rely on resources to complete it. For details, please see *Basic Concepts>Resource*.

S

- SDK

  Software Development Kits. Generally, it is a collection of development tools used by software engineers to build application software for specific software packages, software instances, software frameworks, hardware platforms, operating systems, document packages, etc. MaxCompute currently supports *Java SDK* and Python SDK.

- Authorization

  The project space administrator or project owner gives you permission to perform certain operations on Objects (or objects, such as tables, tasks, resources, etc.) in MaxCompute, including reading, writing, viewing, etc. For the specific operation of authorization, see *User management*.

- Sandbox

  Security restrictions: MaxCompute MapReduce and UDF programs are restricted by Java Sandbox while running in a distributed environment.

T

- Table (table)

  The table is the data storage unit of MaxCompute. See *Basic Concepts>Table*.

- Tunnel

  MaxCompute's data channels provide high concurrency offline data upload and download service. You can use the tunnel service to bulk upload or download data to MaxCompute . Please refer to the tunnel command operation or bulk data channel SDK for relevant commands.

U

- UDF

  Generalized UDF, user defined Function, the Java programming interface provided by MaxCompute develops custom functions, for more information, refer to *User 's guide>UDF*.

  In a narrow sense, UDF refers to user-defined scalar function, whose input and output are one-to-one, that is, reading in a row of data and writing out an output value.

- UDAF

  User Defined Aggregation Function, Custom aggregation function, whose input and output are many-to-one, aggregates multiple input records into one output value. It can be combined with the Group By statement in SQL. For details, please see *Java UDF>UDAF*.

- UDTF

  User Defined Table Valued Function,customictablevaluedFunction, the userDefinedTablevaluedFunction, the customictablevaluedFunction, the function of the function called to the function, the set to the the function to the function, the function to the set to the value to the returned the function, the the function of the function of the the function of the function call to the value of the *Java UDF>UDAF*.

1.2 Project

project is the basic unit of operation in MaxCompute. It is similar to the concept of Database or Schema in traditional databases, and sets the boundary for MaxCompute multi-user isolation and access control.

You can have multiple project permissions at the same time and, by granting relevant authorization, users can access the objects of another project in their own project , such as *Table*, *Resource*, *Function* and *Instance*.

To enter a project (in this example, ‘my project’), run the Use Project command, as follows:

```
use my_project -- Use this command to enter the project space named
my_project.
```

After running the preceding command, you can enter a project named "my_project" and all objects in this project can operated. Use Use Project is a command provided by the MaxCompute client. Before giving a detailed introduction to this part, the document will give a brief introduction to the commonly used commands.For more commands, see *Common Commands*.

Note:
A MaxCompute project is the workspace of DataWorks.

1.3 Table

A table is the data storage unit in MaxCompute. A table is a two-dimensional data structure composed of rows and columns. Each row represents a record, and each column represents a field with the same data type. One record can contain one or more columns. The column name and data type comprise the schema of a table.

The operating objects (input, output) of various computing tasks in MaxCompute are tables. You can create a table, delete a table, and import data into a table.

Note:
The data management module from DataWorks allows you to create, organize, and modify data lifecycles for MaxCompute tables and grant management permissions. For more information, see *data management overview*.

MaxCompute v2.0 supports two types of tables: internal tables and external tables.The MaxCompute2.0 version begins to support the external table.

- For internal tables, all data is stored in MaxCompute tables, and the columns in the table can be any of the data types supported by MaxCompute*Data types*.
- For external tables, data is not stored in MaxCompute. Instead, table data can be stored in *OSS* or *OTS*. MaxCompute only records meta information of the table. You can use MaxCompute’s external table to process unstructured data on OSS or Table Store, such as video, audio, genetics, meteorological, and geographic information.

Use of Dual tables：

- Unlike databases such as Oracle, MaxCompute does not automatically create DUAL tables.

- If you are used to using Dual tables as the most original test tables, you can manually use commands `CREATE TABLE IF NOT EXISTS DUAL (DUMMY VARCHAR(1));` to create an empty table named DUAL with only one field for testing.

  Note:
  At present, the Set commands supported by MaxCompute SQL and new version Mapreduce are divided into two ways:

  - Session Level: To use the new data types (Tinyint, Smallint, Int, Float, Varchar, TIMESTAMP BINARY), add a set statement before the table statement

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

    And submit the execution together with the table statement.
  - Project level: that is to support new types of project level open. The Owner of project can set up project as required.

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

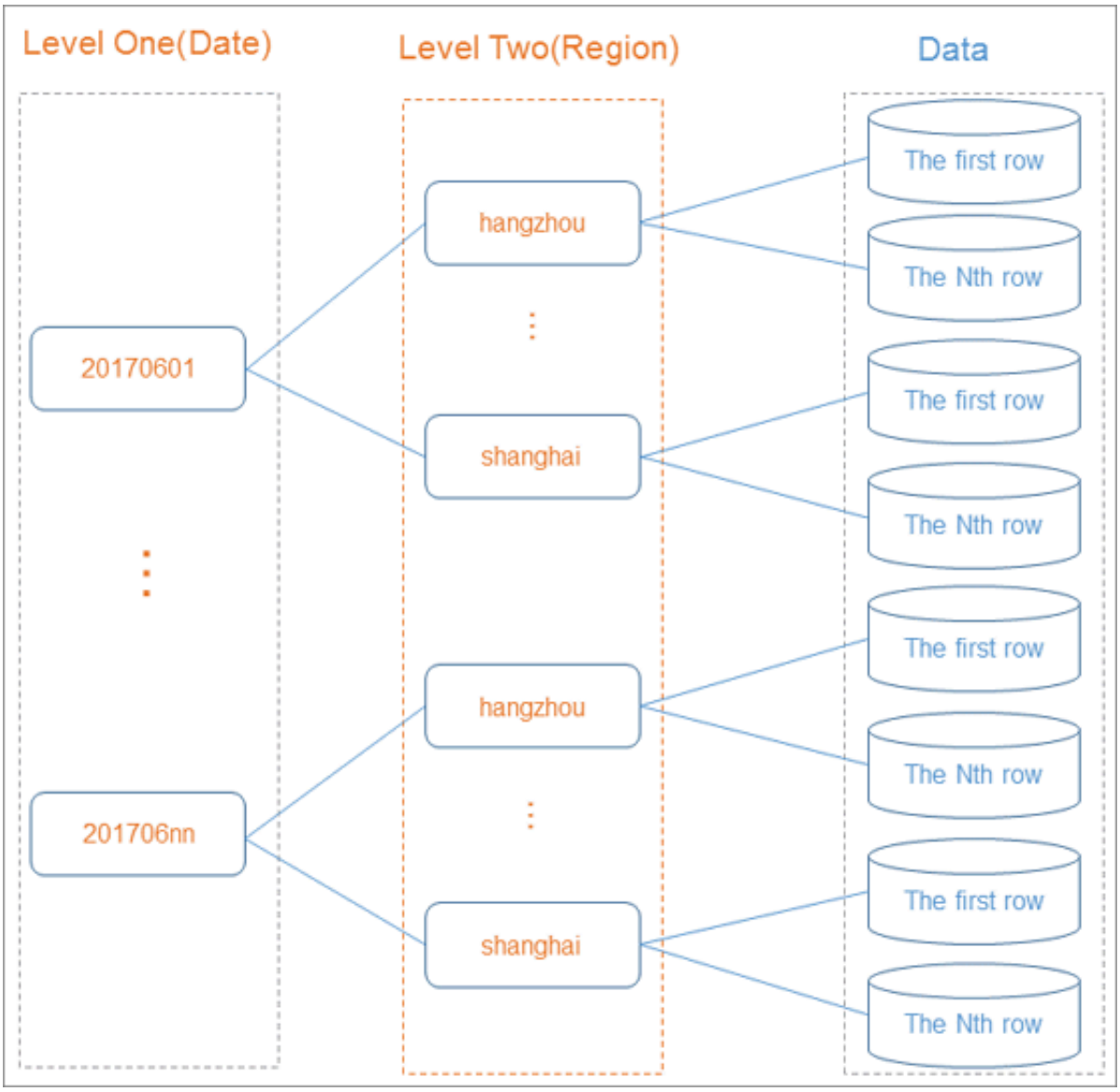
- DUAL is used in the same way as Oracle, such as `select getdate() from dual;`

1.4 Partition

To improve MaxCompute's processing efficiency, you can specify a partition when creating a table. Specifically, several fields in the table can be specified as partition columns. A partition is comparable in terms of functionality to a directory under a file system.

In MaxCompute, each field can be specified as a separate partition. Alternatively, you can specify multiple fields of the table as a partition whereby they function similarly to multi-level directories.

If the partitions to be accessed are specified when you use data, then only corresponding partitions are read and a full table scan is avoided, improving processing efficiency while reducing costs.



Partition types

MaxCompute 2.0 extends the support for partitioning types, currently, MaxCompute supports tinyint, smallint, Int, bigint, varchar, and string partition types.

Note:

In MaxCompute versions earlier than 2.0, only STRING partition type is supported. For historical reasons, although you can specify a partition type of bigint, however, except for the schema representation of the table as bigint, any other case is actually handled as a string. Examples:

```
create table parttest (a bigint) partitioned by (pt bigint);
insert into parttest partition(pt) select 1, 2 from dual;
insert into parttest partition(pt) select 1, 10 from dual;
```

```
select * from parttest where pt >= '2';
```

After the execution, the returned result is only one line, because 10 was treated as a STRING and compared with 2, meaning no result can be returned.

Restrictions

When using a partition, the following restrictions apply:

- The maximum number of partition levels for a single table is 6 levels.
- The maximum number of single table partitions is 60,000.
- The maximum number of query partitions for a query is 10,000.

Examples:

```
-- create a two-level partition table with the date as the level one
partition and the region as the level two partition
create table src (key string, value bigint) partitioned by (pt string,
region string);
```

When querying, use the partition column as a filter condition in the WHERE condition filter:

```
select * from src where pt='20170601' and region='hangzhou';  -- This
example is the correct method of using WHERE conditional filter. When
MaxCompute generates a query plan, only data of the region 'hangzhou'
under the '20170601' partition is accessed.
select * from src where pt = 20170601;   -- This example is an
incorrect method of using the WHERE conditional filter. In this
 example, the effectiveness of the partition filter cannot be
guaranteed. pt is a STRING type. When the STRING type is compared with
 BIGINT type (20170601), MaxCompute converts both to DOUBLE type, and
loss of precision occurs.
```

Some SQL operations on the partitions are less efficient and may cause higher billing, for example, *using dynamic partition*.

For some of the operation commands for MaxCompute, there is a difference in the syntax when you process the partition and non-partition tables, for more information, see the *DDL statement* and using dynamic partitioning.

1.5 Data types

This article introduces you to the data types supported by MaxCompute 2.0, including basic data types and complex types.

Basic data types

supported by MaxCompute2.0 are listed in the following table. Columns in a MaxCompute table must be any of the listed types. New types include TINYINT, SMALLINT, INT, FLOAT, VARCHAR, TIMESTAMP, and BINARY data type.The details are as follows:

Note:

At present, the Set commands supported by MaxCompute SQL and new version Mapreduce are divided into two ways:

- Session Level: To use the new data types (Tinyint, Smallint, Int, Float, Varchar, TIMESTAMP and BINARY), add a set statement before the table statement

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

  And submit the execution together with the table statement.The SQL tool submitted through MaxCompute Studio automatically adds the set statement to submit execution by default.
- When it comes to INT types, 32 bits are added to the set statement and converted to BIGINT, 64 bits, if not added.
- SDK 0.27.2-public version and above, Client 0.27.0 version and above support new data type.

Open new type`odps.sql.type.system.odps2`of influence.

- Implicit type conversion rules change.

  Possible compatibility issues: Some implicit type conversions under the new type system will be disabled, including string - > bigint, string - datetime, double - > bigint, decimal - > double, decimal - > bigint are all at risk of precision loss or error. This situation can be solved by cast transformation.
- Support operations, built-in functions, UDF is not the same.

  Some operations and built-in functions that take new types as parameters and return values are not available without opening new types, including UDF, which overloads new types as parameters and return values. Possible compatibility issues: For example, UDF contains two overloads: bigint and int. The old type must be the overload of bigint, while the new type may be resolved to the overload of int.

- The type of constant will change.

  A single shaping constant, such as 123, is bigint type under the old type, and int type under the new type. Possible compatibility issues: Type int leads to inconsistencies in function prototypes invoked by subsequent operations, including new type tables generated after dropping (i.e., writing to disk) that lead to changes in the behavior of peripheral tools and subsequent jobs.
- The resolution of the bigint keyword is different.

  In the old type system, the int keyword is treated as bigint, while the new type system is treated as real int. Possible compatibility issues: Type int leads to inconsistencies in function prototypes invoked by subsequent operations, including new type tables generated after dropping (i.e., writing to disk) that lead to changes in the behavior of peripheral tools and subsequent jobs.

For a detailed description of setproject, please see:*Other operations*

.

MR type tasks do not support new data types for the time being.

Type	New	Constant	Description
TINYINT	Yes	1Y , -127Y	8-bit signed integer, range -128 to 127
SMALLINT	Yes	32767S, -100S	16-bit signed integer, range -32768 to 32767
INT	Yes	1000,-15645787 (Note2)	32 bit signed shaping, the range is -2^{31} to 2^{31} - 1.
BIGINT	No	100000000000L, -1L	64 bit signed shaping, the range is -2^{63} + 1 to 2^{63} - 1.
FLOAT	Yes	None	32-bit binary floating point
DOUBLE	No	3.1415926 1E+7	64-bit binary floating point
DECIMAL	No	3.5BD , 99999999999 .9999999BD(Note1)	Decimal precision number type, shaping part range -10^{36} + 1 to 10^{36} - 1, decimal part accurate to 10^{-18}

Type	New	Constant	Description
VARCHAR	Yes	None (Note3)	Variable-length character type, n is the length, and the range is 1 to 65535.
STRING	No	“abc”,’bcd’,”alibaba” ‘ inc’ (Note4)	A single string length can be up to 8M
BINARY	Yes	None	Binary data type, a single string length can be up to 8M
DATETIME	No	DATETIME ‘2017-11-11 00:00:00’	Date-time type, range from December 31 , 999 to January 1 -9, 0000, exact to milliseconds (note 5)
TIMESTAMP	Yes	TIMESTAMP ‘2017 -11-11 00:00:00. 123456789’	It interpreted to be timezoneless and ranges from January 1st 0000 to December 31, 9999 23.59: 59.999999999, and is accurate to nanosecond-level.
BOOLEAN	No	TRUE , FALSE	True/False, Boolean type

All data types in the preceding table can be NULL.

Note:

- NOTE 1:When insert is constant to decimal field, pay attention to the writing of constants. For example,3.5BD in the definition of constants.

```
insert into test_tb(a) values (3.5BD);
```

  The a field is decimal type.
- NOTE 2: For INT constant, if the range of INT is exceeded, INT is converted into BIGINT; if the range of BIGINT is exceeded, it is converted into DOUBLE.

In MaxCompute versions earlier than 2.0, all INT types in SQL script are converted to BIGINT , for example:

```
create table a_bigint_table(a int); -- the int here is actually
treated as a bigint
select cast(id as int) from mytable; -- the int here is actually
treated as a bigint
```

To be compatible with earlier MaxCompute versions, MaxCompute 2.0 retains this conversion without setting odps.sql.type.system.odps2 as True. However, a warning is triggered when INT is treated as BIGINT. In this case, we recommend that you change an Int to a Bigint to avoid confusion.

- NOTE 3: VARCHAR constants can be expressed by STRING constants of implicit transforma tion.
- NOTE 4: STRING constants support connections, for example, `abc xyz` is parsed as `abcxyz`, and different parts can be written on different lines.
- NOTE 5: The time value displayed by the current query does not contain milliseconds. The tunnel command specifies the time format through `-dfp` , and can be specified in milliseconds, such as `tunnel upload -dfp 'yyyy-MM-dd HH:mm:ss.SSS'` , for more information about tunnel commands, refer to *Tunnel commands*.

Complex data types

MaxCompute2.0 supports the complex data types listed in the following table.

Type	Definition method	Construction method
ARRAY	array< int >;array< struct< a:int , b:string >>	array(1, 2, 3); array(array(1, 2); array(3, 4))
MAP	map< string, string >;map< smallint, array< string>>	map(“k1”, “v1”, “k2”, “v2”); map (1S, array(‘a’, ‘b’), 2S, array(‘x ’, ‘y))
STRUCT	struct< x:int, y:int>;struct< field1:bigint, field2:array< int>, field3:map< int, int>>	named_struct(‘x’, 1, ‘y’, 2); named_struct(‘field1’, 100L, ‘ field2’, array(1, 2), ‘field3’, map (1, 100, 2, 200)

Note:

At present, the Set commands supported by MaxCompute SQL and new version Mapreduce are divided into two ways:

- Session Level: To use the new data types (Tinyint, Smallint, Int, Float, Varchar, TIMESTAMP BINARY), add a set statement before the table statement

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

  And submit the execution together with the table statement.
- Project level: that is to support new types of project level open. The Owner of project can set up project as required.

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

  For a detailed description of setproject, please see:*Other operations*.

1.6 Lifecycle

This article introduces the concept of life cycle of MaxCompute tables in detail.

The lifecycle of a MaxCompute table or partition is measured from the last update time. If the table or partition remains unchanged after a specified time, MaxCompute automatically recycles it. The **specified time** indicates the lifecycle.

- Lifecycle units: days, positive integers only.
- When a lifecycle is specified for a non-partition table, the lifecycle is counted from the last time the table data was modified (LastDataModifiedTime). If table data has not been changed, MaxCompute recycles the table automatically without manual operation (similar to the drop table operation).
- When a lifecycle is specified for a partition table, we will decide whether a partition should be recycled according to each partition's LastDataModifiedTime. Unlike non-partition tables, a partition table will not be deleted even if its last partition has been recycled.

  Note:

  Lifecycle scanning is started at a scheduled time every day, and entire partitions are scanned. If the partition remains unchanged after its lifecycle, MaxCompute automatically recycles it.

  When a lifecycle is specified for a partition table, MaxCompute determines whether to recycle the partition based on its LastDataModifiedTime. Unlike non-partition tables, a partition table cannot be deleted even when all its partitions have been recycled.
- You can set the lifecycle of tables, but not of partitions. The lifecycle of a table can be specified during table creation.

- If no lifecycle is specified, the table, or partition cannot be automatically recycled by MaxCompute.

For more information on specifying or modifying lifecycles during table creation, and modifying a table's LastDataModifiedTime, see *Table Operations*.

1.7 Resource

This paper introduces the concept of MaxCompute Resource, which can provide resource dependency for MaxCompute specific operations.

The concept of resources

Resources is a concept that is unique to MaxCompute. To accomplish tasks using user-defined functions (for more information, see *UDF*), or *MapReduce*, you must use resources.

- **SQL UDF**: After writing a UDF, you must compile it as a Jar package and upload the package to MaxCompute as a resource. Then, when you run this UDF, MaxCompute automatically downloads its corresponding JAR package to obtain the written code. The JAR package is one type of MaxCompute resource.
- **MapReduce**: After writing a MapReduce program, you must compile it as a Jar package and upload the package to MaxCompute as a resource. Then, when running a MapReduce job, the MapReduce framework automatically downloads the corresponding JAR package and obtain the written code. You can upload text files and MaxCompute tables to MaxCompute as different types of resources. Then, you can read or use these resources when running UDF or MapReduce.

Resource type

MaxCompute provides interfaces for you to read and use resources. For more information, see *Use Resourse Example* and *UDTF Usage* .

Note:

For more information about the resource reading capabilities of user-defined functions (*UDF*) or *MapReduce* , see *Application Restriction*.

The max size that MaxCompute support for single resource is 500MB. Types of MaxCompute resources include:

- File
- Table: tables in MaxCompute

> **Note:**
> Currently, only BIGINT, DOUBLE, STRING, DATETIME, and BOOLEAN fields are supported in tables referenced by MapReduce.

- Jar type, which is compiled Java JAR packages
- Archive type, which is the compression type, and is determined by the resource name suffix. Supported compression types include: .zip/.tgz/.tar.gz/.tar/jar

For more information about resources, see *Add Resource* , *Drop Resource* , *List Resources* and *Describe Resource*.

1.8 Function

This article gives you an overview of the functions that will be used in MaxCompute to provide you with computational functions.

MaxCompute provides SQL computing capabilities. In MaxCompute SQL, you can use the *system's built-in functions* to perform common computing and counting tasks. If the built-in functions do not meet your requirements, you can use the Java programming interface provided by MaxCompute to develop user-defined functions (UDFs).

UDFs can be divided into scalar valued functions, user-defined aggregate functions (UDAFs), and user-defined tables functions (UDTFs).

After writing the code for a UDF, you must compile the code into a JAR package and upload this package to MaxCompute. Then, you can register the UDF in MaxCompute.

> **Note:**
> UDFs are used in the same way as built-in functions, in that you specify the UDF name and input relevant parameters in SQL.

For more information, see *Function introduction*.

1.9 Task

A task is the basic computing unit of MaxCompute. Computing tasks such as those involving SQL, DML and MapReduce functions are completed using tasks.

For most user-submitted tasks, such as *SQL DML statement* , *MapReduce*, etc.MaxCompute first analyzes them and then generates a task execution plan. The execution plan is composed of

multiple execution stages that are dependent on each other. An execution plan consists of multiple stages with dependency links.

Currently, an execution plan can be logically viewed as a directed graph whose vertices represent the stages and whose edges represent the dependency links of the stages. MaxCompute executes each stage according to the dependencies in the graph (execution plan). A single stage comprises multiple threads, also known as workers. These workers complete the computing in this stage. Different workers in the same stage have exactly the same execution logic, but they process different data. Computational tasks are executed directly in MaxCompute instances, for example, *Status Instance* and *Kill Instance*.

For MaxCompute tasks that are not computational tasks, such as DDL statement in SQL, these tasks can only read and modify the metadata information in MaxCompute. This means that no execution plan can be analyzed and generated from the task.

Note:

Not all the requests are converted into tasks in MaxCompute, for example, the operations of *Project*, *Resource*, *UDF* and *Instance* can be completed without MaxCompute tasks.

1.10 Instance

This article introduces you to the MaxCompute task instance and its status.

In MaxCompute, most *tasks* are initiated in MaxCompute instances. MaxCompute instances can be in one of two phases: Running and Terminated.

The status of the running phase is ‘Running’, while the status of the Terminated phase can be ‘Success’, ‘Failed’ or ‘Canceled’. You can query or change the status using the instance ID assigned by MaxCompute. For example:

```
status <instance_id>; --View the status of a certain instance.
            kill <instance_id>; --Stop an instance and set its status
as ‘Canceled’.
            wait <instance_id>; --View the running logs of a certain
instance.
```

2 Common commands

2.1 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 a client*Quick Start.
- For more information about the SDK, see *MaxCompute SDK introduction*MaxCompute SDK introduction.

List of common commands

add alias alter

cost create

delete/drop desc/describe download/get

extended

flag/flags function functions

get

help history

instance/instances

jar/mapreduce

kill

lifecycle list

odpscmd

partition

q/quit

resource

set show sql stop/kill

tunnel

upload

wait who

Use limits

- When you perform resource operations, please note that the size of each resource file should not exceed 500 M, and the total size of resources referenced by a single SQL or MapReduce task should not exceed 2048M. For more restrictions, see*MR limits*。

2.2 project operations

This article introduces you to command operations for entering the project and setting space properties (permissions and whitelist functions, etc.).

Enter the workspce

Command format:

```
use <project_name>;
```

Action:

- Enter the specified workspce. After entering the workspce, all objects in this workspce can be operated by the user.
- If the workspce does not exist or the current user is not in this workspce, an exception is returned.

Example:

```
odps:my_project>use my_project; --my_project is a workspce the user
has privilege to access.
```

Note:

The preceding examples uses the MaxCompute client. All MaxCompute command keywords, workspce names, table names, column names are case insensitive.

Creating a project is creating a MaxCompute project

After running the command, you can access the objects of this workspce. In the following example, assume that test_src exists in the project ‘my_project’. Run the following command:

```
odps:my_project>select * from test_src;
```

MaxCompute automatically searches the table in my_project. If the table exists, it returns the data of this table. If the table does not exist, an exception is thrown. To access the table test_src in another workspce, such as ‘my_project2’, through the project ‘my_project’, you must first specify the workspce name as follows:

```
odps:my_project>select * from my_project2.test_src;
```

The returned data is the data in my_project2, not the initial data of test_src in my_project.

MaxCompute does not support commands to create or delete workspce. You can use the MaxCompute console for additional configurations and operations as needed.For details, see *project list*

SetProject

Command format:

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

Action:

- Use **`setproject`** command to set workspce attributes.

The following example sets the method that allows a full table scan.

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

- If the value of <KEY>=<VALUE> is not specified, the current workspce attribute configuration is displayed. **Command format:**

```
setproject;  --Display the parameters set by the setproject command.
```

Parameters

Property name	Configured permission	Description	Value range
odps.sql.allow.fullscan	ProjectOwner	Determines whether to allow a full table scan.	True (permitted) / false (prohibited)
odps.table.drop.ignorenonexistent	All users	Whether to report an error when deleting a table that does not exist. When the value is true, no error is reported.	True (no error reported)/false
odps.security.ip.whitelist	ProjectOwner	Specify an IP whitelist to access the workspce.	IP list separated by commas (,)
odps.instance.remain.days	ProjectOwner	Determines the duration of the retention of the instance information.	[3- 30]
READ_TABLE_MAX_ROW	ProjectOwner	The number of data entries returned by running the Select statement in the client.	[1-10000]

Examples for odps.security.ip.whitelist

MaxCompute supports a workspce level IP whitelist.

Note:

- If the IP whitelist is configured, only the IP (console IP or IP of exit where SDK is located) in the whitelist can access this workspce.
- After setting the IP white list, wait for at least five minutes to let the changes take effect.
- For further assistance, open a ticket to contact Alibaba Cloud technical support team.

The following are the three formats for an IP list in the whitelist, which can appear in the same command. Use commas (,) to separate these commands.

- IP address: For example, 101.132.236.134.
- Subnet mask: For example, 100.116.0.0/16.
- Network segment: For example, 101.132.236.134-101.132.236.144.

Example of the command line tool set the IP white list:

```
setproject odps.security.ip.whitelist=101.132.236.134,100.116.0.0/16,
101.132.236.134-101.132.236.144;
```

If no IP address is added in the whitelist, then the whitelist function is disabled.

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

2.3 Table operations

This article will show you how to create, delete and view tables using common commands through the MaxCompute client.

If you want to operate a table, use common commands in the client. Moreover it is more convenient to collect tables, apply permissions, and view partitions using the visible data table manager in DataWorks. For more information, see *Table Details*.

Create tables

Command format:

```
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
```

Action:

Create a table.

Note:

- Both the table name and column name are case insensitive and follow the same naming conventions. The name can be up to 128 bytes in length and can contain letters, numbers, and underscores (_).
- The comment content is an effective string, and it can be up to 1,024 bytes in length.

- [LIFECYCLE days]: The parameter ‘days’ refers to the time required to complete a ‘Table Operation’ lifecycle. It must be a positive integer. The unit is ‘day’.
- Suppose that the ‘table_name’ is no-partition table. If calculated from the last updated date, the data is still not modified after N (days), then MaxCompute automatically recycles the table without user intervention (similar to ‘drop table’ operation).
- Suppose that the ‘table_name’ is a partition table. MaxCompute determines whether to recycle the table according to LastDataModifiedTime of each partition. Unlike non-partitioned tables, a partitioned table is not deleted after all its partitions are reclaimed. The lifecycle can only be created for tables and not for the specified partitions.

Example:

```
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 sale_detail.
```

Drop Table

Command format:

```
DROP TABLE [IF EXISTS] table_name; -- Table name to be deleted.
```

Action:

- Delete a table.
- If the option [IF EXISTS] is specified, regardless of whether the table exists or not, the return is successful. If the option [IF EXISTS] is not specified, and the table does not exist, an exception is returned.

Example:

```
DROP TABLE sale_detail; -- If the table exists, success returns.
DROP TABLE IF EXISTS sale_detail; -- No matter whether the table
sale_detail exists or not, success returns.
```

Describe Table

Command format:

```
DESC <table_name>; -- Table name or view name.
DESC extended <table_name>; -- View the extended table information.
```

Action:

Return information of a specified table, includes:

- Owner: The owner of the table.
- Project: The project to which a table belongs.
- CreateTime: The creation time of the table.
- LastDDLTime: The last DDL operation.
- LastModifiedTime: The last time of table modification.
- InternalTable: Indicates the object to be described is table. The value is 'YES' by default.
- Size: Storage size occupied by table data, usually the compression ratio is 5. The unit is Byte.
- Native Columns: Non-partition column information, including column name, type, comment.
- Partition Columns: Partition column information, including partition name, type, and comment.
- Extended Info: The information of extended table, such as StorageHandler and Location.

Example:

```
odps@ project_name>DESC sale_detail; -- Describe a partition table.
+------------------------------------------------------------------------------------
+
| 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 | |
+------------------------------------------------------------------------------------
+
```

Note:

- The preceding example is executed using the MaxCompute client.
- If the table has no partition, the information of Partition Columns is not displayed.

- To describe a 'View', the option ‘InternalTable’ cannot be displayed but the option ‘VirtualView’ can be displayed and its value is ‘YES’ by default. Similarly, the 'Size' option is replaced by the 'View Text' option, which represents the definition of the view, for example: `select * from src`. For more information, see *View operations*.

View partition table

Command format:

```
desc table_name partition(pt_spec
```

Action:

View the specific partition information of a partition table.

Example:

```
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
```

Show Tables/Show Tables like

Command format:

```
SHOW TABLES;
SHOW TABLES like 'chart';
```

Action:

- SHOW TABLES: List all tables of current project.
- SHOW TABLES like 'chart': Lists the tables on which the following table names of the current project match the 'chart'. Regular expressions are supported.

Example:

```
odps@ project_name>show tables;
odps@ project_name>show tables like 'ods_brand*';
ALIYUN$odps_user@aliyun.com:table_name
```

```
......
```

Note:

- The preceding example is executed using the MaxCompute client.
- ALIYUN is a system prompt, indicating the you are an Alibaba Cloud user.
- In this example,odps_user@aliyun.com is the creator of the table in this example.
- In this example,table_name is the name of the table.

Show Partitions

Command format:

```
SHOW PARTITIONS ;  -- table_name: Specify the table to be queried. If
the table does not exist or it is not a partition table, an exception
is thrown.
```

Action:

List all partitions of a table.

Example:

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

Note:

- The preceding example is executed using the MaxCompute client.
- Partition_col1 and partition_col2 are the partition columns of the table.
- Col1_value1, col2_value1, col1_value2, and col2_value2 are corresponding values of the partition columns.

2.4 Instances

Show instances/Show P

Command format:

```
SHOW INSTANCES [FROM startdate TO enddate] [number];
SHOW P [FROM startdate TO enddate] [number];
SHOW INSTANCES [-all];
SHOW P [-all];
```

Action:

Displays the information about the instances created by the current users.

Parameters:

- **`startdate`、`enddate`**: Returns the instance information during the specified period (from startdate to enddate) in the yyyy-mm-dd format and the unit is ‘day’. The parameters are optional. If the parameters are not specified, instances submitted within three days are returned by default.
- **`number`**: Specifies the number of instances to be displayed. Based on the scheduled time, return N (number) instances nearest to the current time. If not specified, all instances that meet the requirements are shown.
- **`-all`**: The information of all instances that meet requirements is returned. To execute the command, you must have the 'list' permission for the project. This command can only return 50 records by default. You can **`-limit number`** to show more record. For example, use `show p -all -limit 100` to show 100 instance records in the project.
- The output items: Include StartTime (the time accurate to seconds), RunTime (s) and Status (including Waiting, Success, Failed, Running, Cancelled, and Suspended).

InstanceID and corresponding SQL are as follows:

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

The probable status of an instance is as follows:

- Running
- Success
- Waiting
- Failed (job failed but data in the target table is modified)
- Suspended
- Canceled

Note:

The commands from the preceding example run in MaxCompute client.

Status Instance

Command format:

```
status <instance_id>; -- instance_id: the unique identifier of an
instance, to specify which instance to be queried.
```

Action:

- Query the status of specified instance, such as Success, Failed, Running, and Cancelled.
- If this instance is not created by the current user, exception is returned.

Example:

```
odps@ $project_name>status 20131225123xxxxxxxxxxxxxx;
Success
```

Query the status of an instance which ID is 20131225123xxxxxxxxxxxxxx, and the result is Success.

Note:

The commands from the preceding example run in MaxCompute client.

Top Instance

Command format:

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

Action:

Permission requirements: The user must be a project owner or administrator.

`top instance`: Displays the job information of the current account that is running in the project. It is displayed, includesding ISNTANCEID , Owner, Type, StartTime, Progress, Status, Priority, RuntimeUsage (CPU/MEM), TotalUsage (CPU/MEM), QueueingInfo (POS/LEN) and so on.

`top instance-all` : Returns all jobs that are currently being executed in the current project. This command can only return 50 records by default. You can user **`-limit number`** to show more record.

Example:

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

Note:

The commands from the preceding example run in MaxCompute client (version 0.29.0 or later).

Kill Instance

Command format:

```
kill <instance_id>; -- instance_id: The unique identifier of an
instance,  which must be ID of an instance whose status is 'Running',
otherwise, an error is returned.
```

Action:

Stop specified instance. The instance must be in the Running status.

Example:

```
odps@ $project_name>kill 20131225123xxxxxxxxxxxxxx;
```

Stop the instance which ID is 20131225123xxxxxxxxxxxxxx.

Note:

- The commands from the preceding example run in MaxCompute client.
- This is an asynchronous process. It does not mean that the distributed task has stopped after the system accepts the request and returns the result. You can check whether the instance is deleted by using the **`status`** command.

Desc Instance

Command format:

```
desc instance <instance_id>; -- instance_id: The unique identifier of
an instance.
```

Action:

Get the job information according to instance ID, including SQL, owner, startime, endtime, status.

Example:

```
odps@ $project_name> desc instance 20150715xxxxxxxxxxxxxx;
ID 20150715xxxxxxxxxxxxxx
Owner ALIYUN$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;
```

Query all the job information related to the instance whose ID is 20150715xxxxxxxxxxxxxx.

Note:

The commands from the preceding example run in MaxCompute client.

Wait instance

Command format:

```
wait <instance_id>;  -- instance_id: The unique identifier of an
instance.
```

Action:

Get running task information, including logs based on the instance ID and a logview link. View task details by accessing the logview link.

Example:

```
wait 201709251611xxxxxxxxxxxxx;
ID = 201709251611xxxxxxxxxxxxx
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.
com/xxxxxxxxxx
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)
```

2.5 Resources

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

You can also search and upload resources using the visualized online data development tools in DataWorks. For more information, see *Resource management*.

Add a resource

Command format:

```
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 jar <local_file.jar> [comment 'cmt'][-f];
```

Parameters

- **`file/archive/table/jar`**: Indicates the resource type. For more information, see *Resources*.
- **`local_file`**: Indicates path of the local file, and uses this file name as the resource name. Resource name also acts as a unique identifier of a resource.
- **`table_name`**: Indicates table name in MaxCompute. Currently, external tables cannot be added into resource.
- **`[PARTITION (spec)]`**: When the resource to be added is a partition table, MaxCompute only supports taking a partition as a resource, not the entire partition 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 function.
- **`[comment 'cmt']`**: Adds a comment for the resource.
- **`[-f]`**: If a name is duplicated, this parameter can be added as a substitute to the original resource. If this parameter is not specified and the duplicate resource name exists, the operation fails.

Example

```
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.
```

```
---Add a resource named sale.res in MaxCompute.
```

Note:

Each resource file size cannot exceed 500 MB. The resource size referenced by a single SQL or MapReduce task cannot exceed 2048 MB. For more information about, see *MR Restrictions* .

Delete a resource

Command format:

```
DROP RESOURCE <resource_name>; --resource_name：a specified resource
name.
```

View the resource list

Command format:

```
LIST RESOURCES;
```

Action:

View 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

Use the following command format to download resources:

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

Action:

Download resources to your local device. The resource type must be file, jar, archive, or py.

Example:

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

```
OK
```

2.6 Functions

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. For more information, see *Function Management*.

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 test_lower AS org.alidata.odps.udf.examples.Lower
USING my_lower.jar;
```

```
USING 'my_lower.jar';
```

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

```
 create function test_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.UDTFResour
ce 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'.
```

2.7 Set operation

This article shows you how to use the set command to set maxcompute or a user-defined system variable, and how to clear the set command settings.

Set

Command Format:

```
Set <key> = <value>
```

Actioin:

You can use the set command to set MaxCompute or affect the MaxCompute operation.

Currently, the system variables supported in MaxCompute are as follow:

```
--Set commands supported by MaxCompute SQL and Mapreduce (new version)
set odps.sql.allow.fullscan=  --Set whether to allow a full table scan
 on a partitioned table. True means allow, and false means not allow.
set odps.stage.mapper.mem=   --Set the memory size of each map worker
. Unit is M and default value is 1024M.
set odps.stage.reducer.mem= --Set the memory size for each reduce
worker in the unit of M. The default value is 1,024M.
set odps.stage.joiner.mem=   --Set the memory size of each join worker
. Unit is M and default value is 1024M.
set odps.stage.mem =    --Set the memory size of all workers in
MaxCompute specified job. The priority is lower than preceding three
set key. Unit is M and no default value.
set odps.stage.mapper.split.size=   -- Modify the input data quantity
of each map worker; that is the size of input file burst. Thus control
 the worker number of each map stage. Unit is M and the default value
is 256M.
set odps.stage.reducer.num=    --Modify the worker number of each
reduce stage and no default value.
set odps.stage.joiner.num=    --Modify the worker number of each join
stage and no default value.
set odps.stage.num=    --Modify the worker concurrency of all stages
 in MaxCompute specified job. The priority is lower than preceding
three set key and no default value.
set odps.sql.type.system.odps2=    --The default value is false. You
must set true when there are new data types such as TINYINT, SMALLINT
, INT, FLOAT, VARCHAR, TIMESTAMP, and BINARY in SQL statement.
```

Show Flags

Command Format:

```
show flags;  --Display the parameters set by the Set command.
```

Action:

Running the **`Use Project`** command can clear the configurations set by the Set command.

2.8 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 = 20150715113033121xxxxxxxx
Input:65727592 Bytes
UDF:0
Complexity:1.0
```

3 Data upload and download

3.1 Data upload and download

This article provides a brief introduction about the upload and download process of the MaxCompute system data, including service connection, SDKs, tools, and cloud data migration.

The DataHub and Tunnel offers the real-time data tunnel and the batch data tunnel respectively to access the MaxCompute system.

Both DataHub and Tunnel provide their own SDKs. The SDKs and derivative data upload and download tools can suffice your data upload and download requirements in various scenarios.

Data upload and download tools include: DataWorks, DTS, OGG plugin, Sqoop, Flume plugin, Logstash plugin, Fluentd plugin, Kettle plugin, MaxCompute console.

Underlying data tunnels used by these tools include:

- DataHub tunnel tools
  - OGG
  - Flume
  - LogStash
  - Fluentd
- Tunnel tools
  - DataWorks
  - DTS
  - Sqoop
  - Kettle
  - MaxCompute console

A wide range of data upload and download tools are applicable to most of the cloud data migration scenarios. The subsequent articles introduce the tools, Hadoop data migration, database data synchronization, log collection, and other cloud migration scenarios. We recommend that you refer to these articles when you select the technical solutions.

3.2 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.

3.3 Cloud data migration

Data upload and data download tools of the MaxCompute platform can be used for a wide range of cloud data migration scenarios. This article introduces some typical scenarios.

Hadoop data migration

For Hadoop data migration, either use Sqoop or DataWorks.

- Sqoop runs an MR job on the original Hadoop cluster for the distributed data transmission to MaxCompute and is highly efficient. For more information, see *Sqoop tool introduction*.
- DataWorks can be combined with DataX for Hadoop data migration.

Database synchronization

To synchronize the data of a database to MaxCompute, select an appropriate tool based on the database type and synchronization rule.

- For offline batch data synchronization, use DataWorks. It supports a wide range of database types, including MySQL, SQL Server, and PostgreSQL. For more information, see *Data synchronization introduction*. For instance operation instructions, see *Create a synchronization task*.
- For real-time Oracle data synchronization, use OGG plug-in tools.
- For real-time RDS data synchronization, use DTS.

Log collection

For log collection, use Flume, Fluentd, and Logstash tools.

3.4 Data upload and download tools

The MaxCompute platform supports a wide range of data upload and download tools. The source code for most of the tools can be found on GitHub, the open-source community to upload and download the data. You can select the tool according to the scenario. The tools are divided into two types: Alibaba Cloud DTplus products and open-source products. This article helps you learn more about these tools.

Alibaba Cloud DTplus products

- **Data integration of DataWorks**

  Data Integration, or data synchronization, 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 types: MaxCompute, RDS (MySQL, SQL Server, and PostgreSQL), Oracle, FTP, AnalyticDB (ADS), OSS, Memcache, and DRDS. For more information, see *Data synchronization introduction*, and for methods of use, see *Create a data synchronization task*.

- **MaxCompute console**

  - For information about console installation and basic use, see *Client introduction*.
  - Based on the *Batch data* tunnel SDK, the client provides built-in Tunnel commands for data upload and download. For more information, see *Basic Tunnel command usage*.

  > **Note:**
  > This is an open-source *aliyun-odps-console*.

- **DTS**

  Data Transmission (DTS) is a data service provided by Alibaba Cloud that supports data exchanges between RDBMS, NoSQL, OLAP, and other data sources. It provides data migration, real-time data subscription, real-time data synchronization, and other data transmission features.

DTS supports data synchronization from ApsaraDB for RDS and MySQL instances to MaxCompute tables. Currently, other data source types are not supported.

Open-source products

- **Sqoop**

  As a tool developed based on the Sqoop 1.4.6 community, Sqoop provides enhanced MaxCompute support with the ability to import and export data from MySQL and other relational databases to MaxCompute tables. Data in HDFS/Hive can also be imported to MaxCompute tables.

  Note:

  This is an open-source *aliyun-maxcompute-data-collectors*.

- **Kettle**

  Kettle is an open-source ETL tool based on Java which can run on Windows, Unix, or Linux. It provides graphic interfaces for you to easily define data transmission topology using drag-and-drop components.

  Note:

  This is an open-source *aliyun-maxcompute-data-collectors*.

- **Flume**

  Apache Flume is a distributed and reliable system, which efficiently collects, aggregates, and moves massive volumes of log data from different data sources to a centralized data storage system. It supports multiple Source and Sink plugins.

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

  Note:

  This is an open-source *aliyun-maxcompute-data-collectors*.

- **Fluentd**

  Fluentd is an open-source software product that collects logs, including Application Logs, System Logs, and Access Logs, from various sources. It allows you to select plug-ins to filter and store log data to different data processors, including MySQL, Oracle, MongoDB, Hadoop, and Treasure Data.

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

- **LogStash**

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

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

- **OGG**

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

  Note:
  This is an open-source *aliyun-maxcompute-data-collectors*.

3.5 Tunnel commands

This article introduces you to the instructions for the use of Upload, Show, Resume and other Tunnel upload and download commands.

Features

The *Client* provides *Tunnel* commands for you to use the functions of the original Dship tool.

Tunnel commands are mainly used to upload or download data.

- **`Upload`**: Supports file or directory (level-one) uploading. Data can only be uploaded to a single table or table partition each time. For partitioned tables, the destination partition must be specified.

```
tunnel upload log.txt test_project.test_table/p1="b1",p2="b2";
-- Uploads data in log.txt to the test_project project's test_table
table, partitions: p1="b1",p2="b2".
tunnel upload log.txt test_table --scan=only;
-- Uploads data from log.txt to the test_table table.--The scan
parameter indicates that the data in log.txt must be scanned to
```

```
determine if it complies with the test_table definitions.If it does
not, the system reports an error and the upload is stopped.
```

- **Download**: You can only download data to a single file. Only data in one table or partition can be downloaded to one file each time. For partitioned tables, the source partition must be specified.

```
tunnel download test_project.test_table/p1="b1",p2="b2" test_table.
txt;
-- Download data from the table to the test_table.txt file.
```

- **Resume**: If an error occurs because of network or the Tunnel service, you can resume transmission of the file or directory after interruption. This command allows you to resume the previous data upload operation, but does not support download operations.

```
tunnel resume;
```

- **Show**: Displays the history of the commands used.

```
tunnel show history -n 5;
-- Displays details for the last five data upload/download commands.
tunnel show log;
--Displays the log for the last data upload/download.
```

- **Purge**: Clears the session directory. Use this command to clear history for last three days.

```
tunnel purge 5;
--Clears logs from the previous five days.
```

Tunnel upload and download limits

Tunnel command does not support uploading and downloading data of the Array, Map, and Struct types.

Each session has a 24-hour life cycle on the server. It can be used within 24 hours after being created, and can be shared among processes or threads. The block ID of each session must be unique.

Use of Tunnel commands

Tunnel commands allows you to obtain help information using the Help sub-command on the client. Each command and selection supports short command format.

```
odps@ project_name>tunnel help;
    Usage: tunnel <subcommand> [options] [args]
    Type 'tunnel help <subcommand>' for help on a specific subcommand.
Available subcommands:
    upload (u)
    download (d)
    resume (r)
    show (s)
    purge (p)
    help (h)
```

```
tunnel is a command for uploading data to / downloading data from
MaxCompute.
```

Parameters

- **upload**: Uploads the data to a MaxCompute table.
- **download**: Downloads the data from a MaxCompute table.
- **resume**: If data fails to be uploaded, use the Resume command to resume the upload from where it was interrupted. Do not use this command for download operations. Each data upload or download operation is called as a session. Run the Resume command and specify the session ID to be resumed.
- **show**: Displays the history of the commands used.
- **purge**: Clears the session directory. Use this command to clear history for last three days.
- **help**: Provides 'help' information regarding questions related to Tunnel.

Upload

Import data of local files to MaxCompute tables in the append mode. The sub-commands are used as follows:

```
odps@ project_name>tunnel help upload;
usage: tunnel upload [options] <path> <[project.]table[/partition]>
              upload data from local file
 -acp,-auto-create-partition <ARG> auto create target partition if not
                                   exists, default false
 -bs,-block-size <ARG> block size in MiB, default 100
 -c,-charset <ARG> specify file charset, default ignore.
                                   set ignore to download raw data
 -cp,-compress <ARG> compress, default true
 -dbr,-discard-bad-records <ARG> specify discard bad records
                                   action(true|false), default false
 -dfp,-date-format-pattern <ARG> specify date format pattern, default
                                   yyyy-MM-dd HH:mm:ss;
 -fd,-field-delimiter <ARG> specify field delimiter, support
                                   unicode, eg \u0001. default ","
 -h,-header <ARG> if local file should have table
                                   header, default false
 -mbr,-max-bad-records <ARG> max bad records, default 1000
 -ni,-null-indicator <ARG> specify null indicator string,
                                   default ""(empty string)
 -rd,-record-delimiter <ARG> specify record delimiter, support
                                   unicode, eg \u0001. default "\r\n
"
 -s,-scan <ARG> specify scan file
                                   action(true|false|only), default
true
 -sd,-session-dir <ARG> set session dir, default
                                   D:\software\odpscmd_public\
plugins\ds
                                   hip
 -ss,-strict-schema <ARG> specify strict schema mode. If false,
                                   extra data will be abandoned and
                                   insufficient field will be filled
                                   with null. Default true
```

```
  -te,-tunnel_endpoint <ARG> tunnel endpoint
     -threads <ARG> number of threads, default 1
  -tz,-time-zone <ARG> time zone, default local timezone:
                                       Asia/Shanghai
For example:
    tunnel upload log.txt test_project.test_table/p1="b1",p2="b2"
```

Parameters

- **-acp**: Determines if the operation automatically creates the destination partition if it does not exist. This one is disabled by default.
- **-bs**: Specifies the size of each data block uploaded using Tunnel. Default value: 100MiB (1MiB=1024*1024B) .
- **-c**: Specifies the local data file encoding. Default value: I. When not set, the encoding of the downloaded source data is used by default.
- **-cp**: Determines whether the local file is compressed before being uploaded, reducing traffic usage. It is enabled by default.
- **-dbr**: Determines whether to ignore corrupted data (including extra, missing columns or mismatched column data types).
  - If this value is true, all the data that does not satisfy table definitions is ignored.
  - When the parameter is set to false, the system displays error messages in case of corrupted data, but the raw data in the destination table remains unaffected.
- **-dfp**: Specifies the format of DateTime data. Default value: yyyy-MM-dd HH:mm:ss. If you want to specify the time format to the level of milliseconds, use **tunnel upload -dfp 'yyyy-MM-dd HH:mm:ss.SSS'**, for more information, see *Data types*.
- **-fd**: Specifies the column delimiter of the local data file. The default value is comma (,).
- **-h**: Determines whether the data file contains the header. If it is set to true, Dship skips the header and starts uploading from the next row.
- **-mbr**: By default, if more than 1,000 rows of corrupted data is uploaded, the upload is terminated. This parameter allows you to adjust the tolerated volume of the corrupted data.
- **-ni**: Specifies the NULL data identifier. Default value: “ “(blank string).
- **-rd**: Specifies the row delimiter of the local data file. Default value: \r\n.
- **-s**: Determines whether to scan the local data file. Default value: false.
  - If set to true, the system scans the data first, and then imports the data if the format is correct.
  - If set to false, the system imports the data directly without scanning.
  - If the value is 'only', then only the local data is scanned. No data is imported after scanning.

- **-sd**: Sets the session directory.
- **-te**: Specifies the tunnel endpoint.
- **-threads**: Specifies the number of threads. Default value: 1.
- **-tz**: Specifies the time zone. The default value is the local time zone: Asia/Shanghai.

Example

- Create a destination table:

```
CREATE TABLE IF NOT EXISTS sale_detail(
      shop_name STRING,
      customer_id STRING,
      total_price DOUBLE)
PARTITIONED BY (sale_date STRING,region STRING);
```

- Add a partition:

```
alter table sale_detail add partition (sale_date='201312', region='
hangzhou');
```

- Prepare the data file data.txt with the following content:

```
shop9,97,100
shop10,10,200
shop11,11
```

The data of the third row of this file is not consistent with the definition in Table sale_detail. The three columns are defined by sale_detail, but this row only has two.

- Import data:

```
odps@ project_name>tunnel u d:\data.txt sale_detail/sale_date=201312
,region=hangzhou -s false
Upload session: 20150610163922488087 0a002ec60c
Start upload:d:\data.txt
Total bytes:41 Split input to 1 blocks
2015-06-10 16:39:22 upload block: '1'
ERROR: column mismatch -,expected 3 columns, 2 columns found, please
 check data or delimiter
```

Because data.txt contains corrupted data, data import fails. The system displays the session ID and error message.

- Verify data:

```
odps@ odpstest_ay52c_ay52> select * from sale_detail where sale_date
='201312';
ID = 20150610084135370gyvc61z5
+-----------+-------------+-------------+-----------+--------+
| shop_name | customer_id | total_price | sale_date | region |
+-----------+-------------+-------------+-----------+--------+
+-----------+-------------+-------------+-----------+--------+
```

The data import failed because of dirty data and hence the table is empty.

Show

Displays historical records. The sub-commands are used as follows:

```
odps@ project_name>tunnel help show;
usage: tunnel show history [options]
              show session information
 -n,-number <ARG> lines
For example:
    tunnel show history -n 5
    tunnel show log
```

Parameter

-n: Specifies the number of rows to be displayed.

Example

```
odps@ project_name>tunnel show history;
20150610163922488087 0a002ec60c failed 'u --config-file /D:/console
/conf/odps_config.ini --project odpstest_ay52c_ay52 --endpoint http
://service.odps.aliyun.com/api --id UlVxOHuthHV1QrI1 --key 2m4r3WvTZb
sNJjybVXj0InVke7UkvR d:\data.txt sale_detail/sale_date=201312,region=
hangzhou -s false'
```

Note:

With reference to the preceding example, **20150610163922488087 0a002ec60c** is the session ID of the failed data importing in the previous section.

Resume

Repairs and re-executes historical records (only valid for data uploads). The sub-commands are used as follows:

```
odps@ project_name>tunnel help resume;
usage: tunnel resume [session_id] [-force]
              resume an upload session
 -f,-force force resume
For example:
    tunnel resume
```

Example

Modify the data.txt file as follows:

```
shop9,97,100
shop10,10,200
```

Re-upload the repaired data:

```
odps@ project_name>tunnel resume 20150610163922488087 0a002ec60c --
force;
start resume
```

```
20150610163922488 0870a002ec60c
Upload session: 20150610163922488 0870a002ec60c
Start upload:d:\data.txt
Resume 1 blocks
2015-06-10 16:46:42 upload block: '1'
2015-06-10 16:46:42 upload block complete, blockid=1
upload complete, average speed is 0 KB/s
OK
```

Note:

With reference to the preceding example, **20150610163922488 0870a002ec60c** is session ID.

Verify data:

```
odps@ project_name>select * from sale_detail where sale_date='201312';
 ID = 20150610084801405g0a741z5
 +-----------+-------------+-------------+-----------+--------+
 | shop_name | customer_id | total_price | sale_date | region |
 +-----------+-------------+-------------+-----------+--------+
 | shop9 | 97 | 100.0 | 201312 | hangzhou |
 | shop10 | 10 | 200.0 | 201312 | hangzhou |
 +-----------+-------------+-------------+-----------+--------+
```

Download

The sub-commands are used as follows:

```
odps@ project_name>tunnel help download;
usage: tunnel download [options] <[project.]table[/partition]> <path>
              download data to local file
 -c,-charset <ARG> specify file charset, default ignore.
                                    set ignore to download raw data
 -ci,-columns-index <ARG> specify the columns index(starts from
                                    0) to download, use comma to split
each
                                    index
 -cn,-columns-name <ARG> specify the columns name to download,
                                    use comma to split each name
 -cp,-compress <ARG> compress, default true
 -dfp,-date-format-pattern <ARG> specify date format pattern, default
                                    yyyy-MM-dd HH:mm:ss
 -e,-exponential <ARG> When download double values, use
                                    exponential express if necessary.
                                    Otherwise at most 20 digits will be
                                    reserved. Default false
 -fd,-field-delimiter <ARG> specify field delimiter, support
                                    unicode, eg \u0001. default ","
 -h,-header <ARG> if local file should have table header,
                                    default false
    -limit <ARG> specify the number of records to
                                    download
 -ni,-null-indicator <ARG> specify null indicator string, default
                                    ""(empty string)
 -rd,-record-delimiter <ARG> specify record delimiter, support
                                    unicode, eg \u0001. default "\r\n"
 -sd,-session-dir <ARG> set session dir, default
                                    D:\software\odpscmd_public\plugins\
dshi
                                    p
```

```
 -te,-tunnel_endpoint <ARG> tunnel endpoint
    -threads <ARG> number of threads, default 1
 -tz,-time-zone <ARG> time zone, default local timezone:
                                  Asia/Shanghai
usage: tunnel download [options] instance://<[project/]instance_id> <
path>
              download instance result to local file
 -c,-charset <ARG> specify file charset, default ignore.
                                  set ignore to download raw data
 -ci,-columns-index <ARG> specify the columns index(starts from
                                  0) to download, use comma to split
each
                                  index
 -cn,-columns-name <ARG> specify the columns name to download,
                                  use comma to split each name
 -cp,-compress <ARG> compress, default true
 -dfp,-date-format-pattern <ARG> specify date format pattern, default
                                  yyyy-MM-dd HH:mm:ss
 -e,-exponential <ARG> When download double values, use
                                  exponential express if necessary.
                                  Otherwise at most 20 digits will be
                                  reserved. Default false
 -fd,-field-delimiter <ARG> specify field delimiter, support
                                  unicode, eg \u0001. default ","
 -h,-header <ARG> if local file should have table header,
                                  default false
    -limit <ARG> specify the number of records to
                                  download
 -ni,-null-indicator <ARG> specify null indicator string, default
                                  ""(empty string)
 -rd,-record-delimiter <ARG> specify record delimiter, support
                                  unicode, eg \u0001. default "\r\n"
 -sd,-session-dir <ARG> set session dir, default
                                  D:\software\odpscmd_public\plugins\
dshi
                                  p
 -te,-tunnel_endpoint <ARG> tunnel endpoint
    -threads <ARG> number of threads, default 1
 -tz,-time-zone <ARG> time zone, default local timezone:
                                  Asia/Shanghai
For example:
    tunnel download test_project.test_table/p1="b1",p2="b2" log.txt
    tunnel download instance://test_project/test_instance log.txt
```

Parameters

- **-c**: Specifies the local data file encoding. Default value: Ignore.
- **-ci**: Specifies the column index (starts from 0) for downloading. Separate multiple entries with commas (,).
- **-cn**: Specifies the names of the columns to download. Separate multiple entries with commas (,).
- **-cp**, **-compress**: Determines whether the data is compressed before it is downloaded, reducing traffic usage. It is enabled by default.
- **-dfp**: Specifies the format of DateTime data. Default value: yyyy-MM-dd HH:mm:ss.

- **-e**: When downloading Double type data, use this parameter to express the values as exponential functions. Otherwise, a maximum of 20 digits can be retained.
- **-fd**: Specifies the column delimiter of the local data file. The default value is comma (,).
- **-h**: Determines whether the data file contains the header. If set to 'true', Dship skips the header and starts downloading from the second row.

  Note:

  `-h=true` and `threads>1` cannot be used together.
- **-limit**: Specifies the number of files to be downloaded.
- **-ni**: Specifies the NULL data identifier. Default value: " "(blank string).
- **-rd**: Specifies the row delimiter of the local data file. Default value: \r\n.
- **-sd**: Sets the session directory.
- **-te**: Specifies the tunnel endpoint.
- **-threads**: Specifies the number of threads. Default value: 1.
- **-tz**: Specifies the time zone. The default value is the local time zone: Asia/Shanghai.

Example

Download data to the *result.txt*:

```
$ ./tunnel download sale_detail/sale_date=201312,region=hangzhou
result.txt;
    Download session: 20150610165824528387 0a002ed0b9
    Total records: 2
    2015-06-10 16:58:24 download records: 2
    2015-06-10 16:58:24 file size: 30 bytes
    OK
```

Verify the content of the *result.txt*:

```
shop9,97,100.0
shop10,10,200.0
```

Purge

Purge the session directory. By default, sessions for last three days are purged. The sub-commands are used as follows:

```
odps@ project_name>tunnel help purge;
usage: tunnel purge [n]
               force session history to be purged.([n] days before,
default
               3 days)
For example:
```

```
    tunnel purge 5
```

Data types:

Type	Required
STRING	String type data. The length cannot exceed 8MB.
BOOLEN	Upload values only support true, false, 0, and 1. Only the values true or false (not case-sensitive) are supported for downloading.
BIGINT	Value range: [-9223372036854775807, 9223372036854775807].
DOUBLE	• 16-bit valid. • Uploads support expression in scientific notation. • Supports only numerical expression for downloading. • Max value: 1.7976931348623157E308. • Min value: 4.9E-324. • Positive infinity: Infinity. • Negative infinity: -Infinity.
DATETIME	By default, Datetime data supports the UTC+8 time zone for data upload. Use the command to specify the format pattern for the date in your data.

If you upload DATETIME type data, specify the time and date format. For more information about specific formats, see SimpleDateFormat.

```
"yyyyMMddHHmmss": data format "20140209101000"
"yyyy-MM-dd HH:mm:ss" (default): data format "2014-02-09 10:10:00"
"MM/dd/yyyy": data format "09/01/2014"
```

Example

```
tunnel upload log.txt test_table -dfp "yyyy-MM-dd HH:mm:ss"
```

Null: All data types can be Null.

- By default, a blank string indicates a Null value.
- The parameter **`-null-indicator`** can be used in the command line to specify a Null string.

```
tunnel upload log.txt test_table -ni "NULL"
```

Character encoding: You can specify the character encoding of the file. Default value: UTF-8.

```
tunnel upload log.txt test_table -c "gbk"
```

Delimiter: The Tunnel commands support custom file delimiters. The row delimiter is '-record-delimiter', and the column delimiter is -field-delimiter.

Description:

- Row and column delimiters of multiple characters are supported.
- A column delimiter cannot contain a row delimiter.
- Only the follow escape character delimiters are supported in the command line: \r, \n, and \t.

Example

```
tunnel upload log.txt test_table -fd "||" -rd "\r\n"
```

3.6 Tunnel SDK

3.6.1 Summary

MaxCompute Tunnel is the data tunnel of MaxCompute. It helps in uploading and downloading data to MaxCompute. However, Tunnel only supports table data upload and download.

Based on the Tunnel SDK, MaxCompute offers *Data upload and download tools*.

When using *Maven*, you can search for **odps-sdk-core** in the Maven database to find different versions of Java SDK. The configuration is as follows: SDK (available in different versions).

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

This article describes the main interfaces of Tunnel SDK, which may differ according to the SDK version. See *SDK Java Doc*.

Interface	Description
TableTunnel	The portal class interface to access the MaxCompute Tunnel service. You can access MaxCompute and its Tunnel using the Internet or intranet of Alibaba Cloud. No traffic fee is incurred when you use intranet to download data through MaxCompute Tunnel. The intranet address is only valid for cloud products in the Hangzhou region.
TableTunnel.UploadSession	Indicates a process of uploading data to a MaxCompute table.
TableTunnel.DownloadSession	Indicates a process of downloading data from a MaxCompute table.

Note:

- For more information about the SDK, see *SDK Java Doc*.
- For more information about service connections, see *Access Domains and Data Centers*.

3.6.2 TableTunnel

TableTunnel is an ingress class that accesses the MaxCompute Tunnel service. The TableTunnel.UploadSession interface is a session that uploads data to the MaxCompute table. The TableTunnel.DownloadSession interface is a session that downloads data to the MaxCompute table.

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 partitionSpec);
 public DownloadSession getDownloadSession(String projectName, String
tableName, PartitionSpec partitionSpec, String id);
 public DownloadSession getDownloadSession(String projectName, String
tableName, String id);
 public UploadSession getUploadSession(String projectName, String
tableName, PartitionSpec partitionSpec, String id);
 public UploadSession getUploadSession(String projectName, String
tableName, String id);
 }
```

Parameters

- Lifecycle: It is the TableTunnel life cycle, begins with a TableTunnel instance creation and ends with the completion of the process.
- PublicClassTableTunnel: A method of creating uploading and downloading objects.
- Session: It is a process for uploading and downloading table or a partition. A session consists of one or more HTTP Requests to the Tunnel RESTful API.
- Uploading session: The uploading session of TableTunnel is INSERT INTO semantics, which means that sessions that upload the same table or partition do not interfere with each other. The upload of each session is located in different directories.
- Block ID: The corresponding file name. In an uploading session, each RecordWriter corresponds to an HTTP Request, identified by a block ID and corresponds to a file on the service side.

- RecordWriter: In a session, opening RecordWriter multiple times with the same block ID results in overwriting. The data uploaded by the last RecordWriter calling close() is retained. This feature can be used for retransmissions when block upload fails.

TableTunnel interface implementation process:

1. RecordWriter.write() uploads data to a file in a temporary directory.
2. RecordWriter.close() moves the preceding file from the temporary directory to the data directory.
3. Session.commit() moves all files in the corresponding data directory to the directory where the corresponding table is located, and updates the table meta. Precisely, the data that moves into the table is visible to other MaxCompute tasks (including SQL and MR).

Limits:

- The range of block id is 0 to 20000. The data size uploaded by a single block is limited to 100 GB.
- The session timeout is 24 hours. Split the massive data into multiple sessions, if the transmissi on time is supposed to exceed the threshold that is 24 hours.
- The HTTP Request timeout for RecordWriter is 120 seconds. If no data flows through the HTTP connection is observed within 120 seconds, the service automatically closes the connection.

  Note:
  By default, HTTP has a buffer of 8 KB. Therefore, it is difficult to determine the data flow through an HTTP connection when you call RecordWriter.write() each time. Moreover, TunnelRecordWriter.flush() can forcibly clear the data from the buffer.
- When logs are being written into MaxCompute, the RecordWriter can be easily timed out as the flow of the data is unpredictable. **Note:**
  - We do not recommend using a RecordWriter for all types of data. Because each RecordWrit er corresponds to a file resulting into numerous small files, critically impacting MaxCompute performance.
  - We recommend calling a RecordWriter to write data in a batch when your code cache data size exceeds 64 MB.
- The threshold for RecordReader timeout is 300 seconds.

3.6.3 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 TunnelExce
ption;
        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:
  1. UNKNOWN: The initial value when the server creates a session.
  2. NORMAL: The upload object is created successfully.
  3. CLOSING: The server changes the status to CLOSING when **complete** is called.
  4. CLOSED: The upload is now complete. Precisely, moving the data to the directory where the result table is located.
  5. EXPIRED: The upload session is timed out.
  6. 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**.

3.6.4 DownloadSession

This DownloadSession interface is defined as follows:

```
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 TunnelExce
ption
        public String getId()
        public long getRecordCount()
        public TableSchema getSchema()
        public DownloadSession.Status getStatus()
        public RecordReader openRecordReader(long start, long count)
        public RecordReader openRecordReader(long start, long count,
boolean compress)
    }
```

Parameters:

- **Life cycle**: Begins with the creation of the Download instance and ends with the completion of a download process.
- **Create Download instance**: An instance can be created either by Calling the Constructor or by using the TableTunnel.
  - Request mode: Synchronous.
  - The server creates a session for this download instance and a unique DownloadId is generated. Obtain this ID using the getId on the client.
  - This operation incurs high costs. The server creates an index for the data files. Large files generally take longer time to download.
  - Simultaneously, the server returns the total number of Records and starts multiple concurrent downloads based on this value.
- **Download data**:
  - Request mode: Asynchronous.
  - Call the **`openRecordReader`** method to generate a RecordReader instance. “start” identifies the start position of downloading this record, which cannot be less than zero. “count” specifies the number of records for this download which must be greater than zero.
- **View download**:
  - Request mode: Synchronous.
  - Call **`getStatus`** to obtain the current download status.
- Following is the list of 4 states:

- UNKNOWN: The initial value when the server creates a session.
- NORMAL: The download object is successfully created.
- CLOSED: The download is now complete.
- EXPIRED: The download session is timed out.

3.6.5 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 ofUploadSession 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.EXPONENTIA
L_BACKOFF)
writer = (TunnelBufferedWriter) uploadSession.openBufferedWriter();
writer.setRetryStrategy(retry);
```

3.7 Bulk data channel SDK example

3.7.1 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.

3.7.2 简单下载示例

```
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";
                        //设置tunnelUrl，若需要走内网时必须设置，否则默认公
网。此处给的是华东2经典网络Tunnel Endpoint，其他region可以参考文档《访问域名和数
据中心》。
        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);//tunnelUrl设置
                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;
                        }
                        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();
        }
 }
```

本示例中，为了方便测试，数据通过System.out.println直接打印出来，在实际使用时，您可改写为直接输出到文本文件。

3.7.3 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 HuaDong 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 PartitionS
pec(partition);
                        UploadSession uploadSession = tunnel.
createUploadSession(project,
                                        table, partitionSpec);
                        System.out.println("Session Status is : "
                                        + uploadSession.getStatus().
toString());
                        ExecutorService pool = Executors.newFixedTh
readPool(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*.

3.7.4 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.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 (I! = 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 HuaDong 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.createDownloadSessio
n(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.newFixedTh
readPool(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.getSchema()));
                        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*.

3.7.5 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();
```

3.7.6 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.createUplo
adSession(projectName, tableName);
    UploadThread t1 = new UploadThread(up);
    UploadThread t2 = new UploadThread(up);
    t1.start();
    t2.start();
    t1.join();
    t2.join();
    uploadSession.commit();
  }
```

3.8 Import or export data using the Data Integration

Use *Data Integration* function of DataWorks to create data synchronization tasks and import and export MaxCompute data.

Prerequisites

Before importing or exporting data, complete the required operations first. For more information, see *Prepare an Alibaba Cloud account* and *Purchase and create a project*.

Add MaxCompute data source

Note:

- Only the project administrator can create a data source. Other roles can only view the data source.
- **If the data source you want to add is a current MaxCompute project, skip this operation** . After this project is created and appears as a Data Integration data source, this project is added as a MaxCompute data source named odps_first by default.

Procedure

1. Log on to the *DataWorks console* as an administrator and click **Enter Workspace** from the Actions column of the relevant project in the **Project List**.
2. Select **Data Integration** from the upper navigation pane. Click **Data Source** from the left-side navigation pane.
3. Click **New Source**.Select MaxCompute (ODPS) from the Large Data Storage section.
4. Enter required configurations in the data dialog box.

   Parameters

   - **Name**: Contains letters, numbers, and underscores (_). It must begin with a letter or an underscore (_) , and cannot exceed 60 characters.

- **`Data source description`**: Provides a brief description of the data source, and cannot exceed 80 characters.
- **`Data source type`**: Currently, it is ODPS.
- **`ODPS Endpoint`**: Read-only by default. The value is automatically read from the system configuration.
- **`ODPS Item name`**: Name of the project, helps to identify the corresponding MaxCompute project.
- **`Access ID`**: The Access ID associated with the account of the MaxCompute project owner.
- **`AccessKey`**: The AccessKey associated with the account of the MaxCompute project owner, used in pairs with the Access ID.

5. (Optional). Click **Test Connectivity** to test the connectivity after entering all the required information in the relevant fields.
6. If the connectivity test is successful, click **Save**.

> **Note:**
> For more information about the other data sources configurations, see *data source configuration*.

Import data through Data Integration

Take importing MySQL data to MaxCompute as an example, you can configure a synchronization task using **Wizard Mode** or **Script Mode**.

Configure a synchronization task in Wizard mode

1. Create a Wizard Mode synchronization task.
2. Select the source.

   Select the MySQL data source and the source table “mytest”. The data browsing area is collapsed by default. Click **Next**.
3. Select a Target.

   The target must be a previously created MaxCompute table. You can also create a new table by clicking **Quick Table Creation**.

   Parameters

   - **`Partition information`**: Specify every level of partition. When writing data to a table with three levels of partitions, you must configure the last partition level, for example, pt=20150101, type=1, biz=2. This item is unavailable for non-partitioned tables.
   - **`Data clearing rules`**:

- **`Clear existing data before writing`**: Before data is imported to a table or partition, all data in the table or partition is cleared, which is equivalent to Insert Overwrite.
- **`Retain existing data before writing`**: Existing data is not cleared before new data is imported. Each operation appends new data, which is equivalent to Insert Into.

4. Map the fields.

   Select the mapping between fields. Configure the field mapping relationships. The **Source Table Fields** on the left correspond one to one with the **Target Table Fields** on the right.

5. Control the channel.

   Click **Next** to configure the maximum job rate and dirty data check rules.

   Parameters

   - **`Maximum job rate`**: Determines the highest rate possible for data synchronization jobs. The actual rate of the job may vary with the network environment, database configuration, and other factors.
   - **`Concurrent job count`**: For a single synchronization job, Concurrent job count * Individual job transmission rate = Total job transmission rate.

   When a maximum job rate is specified, how do you select the concurrent job count?

   - If your data source is an online business database, we recommend that you refrain from setting a large value for the concurrent job count to avoid interference with the online database.
   - If you require a high data synchronization rate, we recommend that you select the highest job rate and a large concurrent job count.

6. Preview and store.

   Make sure the configuration of the task is correct, and click **Save**.

Run a synchronization task

Run a synchronization task directly

If system variable parameters are set in the synchronization task, the variable parameter configuration window is displayed during task operation.

After saving the task, click **Run** to run the task immediately. Click **Submit** and the synchronization task will be submitted to the scheduling system of the DataWorks. The scheduling system automatically and periodically runs the task from the second day according to the configuration

attributes. For more information on scheduling configurations, see *Scheduling configuration description*.

Configure a synchronization task in Script mode

Use the following script to configure synchronization tasks. Other configurations and job operation are the same as **Wizard Mode**.

```
{
  "type": "job",
  "version": "1.0",
  "configuration": {
    "reader": {
      "plugin": "mysql",
      "parameter": {
        "datasource": "mysql",
        "where": "",
        "splitPk": "id",
        "connection": [
          {
            "table": [
              "person"
            ],
            "datasource": "mysql"
          }
        ],
        "connectionTable": "person",
        "Column ":[
          "id",
          "name"
        ]
      }
    },
    "writer": {
      "plugin": "odps",
      "parameter": {
        "datasource": "odps_first",
        "table": "a1",
        "truncate": true,
        "partition": "pt=${bdp.system.bizdate}",
        "Column ":[
          "id",
          "col1"
        ]
      }
    },
    "Setting ":{
      "speed": {
        "mbps": "1",
        "concurrent": "1"
      }
    }
  }
}
```

References

- For the Reader configurations about different types of data sources, see *Configure Reader Plug-ins*.

- For the Writer configurations about different types of data sources, see *Configure Writer Plug-ins*.

3.9 Real-time data tunnel of DataHub

DataHub is a MaxCompute service designed to process streaming data. It allows you to subscribe to streaming data and publish the data. You can easily construct analysis programs and applications based on streaming data.

4 SQL

4.1 SQL summary

This article introduces you to MaxCompute SQL keywords, type conversion instructions, partition tables, UNION ALL operations and use restrictions.

SQL summary

MaxCompute SQL is suitable for various scenarios. The massive data (GB, TB, or EB level) must be processed based on an offline batch calculation. It takes several seconds or even minutes to schedule after a job is submitted. Therefore, MaxCompute SQL is preferred for services that process tens of thousands of transactions per second.

The MaxCompute SQL syntax is similar to SQL and can be considered as a subset of standard SQL. However, the MaxCompute SQL must not be confused with a database. It does not have database characteristics including transactions, primary key constraints, indexes, and so on. The maximum size of SQL in MaxCompute is 3 MB.

Reserved words

MaxCompute SQL considers the keywords of SQL statement as reserved words. If you use keywords for name tables, columns, or partitions, you must escape the keywords with the `` symbol, otherwise an error is occurred. Reserved words are case insensitive and the most common words used are as follows: (For a complete reserved word list, see *MaxCompute SQL Reserved Word*).

```
% & && ( ) * +
 - . / ; < <= <>
 =    >    >=    ?    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
```

Type conversion

MaxCompute SQL allows conversion between data types. The conversion methods include explicit type conversion and implicit type conversion. For more information, see *Type Conversion*.

- Explicit conversions: Uses CAST to convert a value type.
- Implicit conversions: MaxCompute automatically performs implicit conversions while running based on the context environment and conversion rules. Implicit conversion scope includes various operators, built-in functions, and so on.

Partitioned table

MaxCompute SQL supports partitioned tables. Specify the partition as it simplifies the operation. For example, improve SQL running efficiency, reduce the cost, and so on. For more information, see *Basic concept>Partition*.

UNION ALL

To be involved in a UNION ALL operation, the data type of columns, column numbers, and column names must be consistent, otherwise an error occurs.

4.2 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 (_').
	`'aaa' like'a_'= TRUE`
	`'aaa' like'a%' = TRUE`
	`'aaa' like'aab'= FALSE`
	`'a%b' like'a\%b'= TRUE`
	`'axb' like 'a\%b'= FALSE`

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 `A [NOT] BETWEEN B AND C`. 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) < 0.000000001
 -- 0.9999999999 and 1.0000000000 have the precision of 10 decimal
digits, while 0.000000001 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, return3. 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
    A or B TRUE or TRUE=TRUE
                    TRUE or FALSE=TRUE
                    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.

4.3 Type conversions

MaxCompute SQL allows conversion between data types. The two conversion methods are explicit type conversion and implicit type conversion.

Explicit conversion

Explicit conversions use CAST to convert a value type to another. The following table lists the types that can be explicitly converted in MaxCompute SQL.

From/To	Bigint	Double	String	Datetime	Boolean	Decimal
Bigint	–	Y	Y	N	N	Y
Double	Y	–	Y	N	N	Y
String	Y	Y	–	Y	N	Y
Datetime	N	N	Y	–	N	N
Boolean	N	N	N	N	–	N
Decimal	Y	Y	Y	N	N	-

Y means can be converted. N means cannot be converted. – means conversion is not required.

Example:

```
select cast(user_id as double) as new_id from user;
select cast('2015-10-01 00:00:00' as datetime) as new_date from user;
```

Note:

- To convert the Double type to the Bigint type, digits after the decimal point are dropped. For example, `cast(1.6 as bigint) = 1`.
- To convert the String type that meets the Double format to the Bigint type, it is converted to the Double type, and then to the Bigint type. The digits after the decimal point are dropped. For example, `cast("1.6" as bigint) = 1`.
- The String type that meets the Bigint format can be converted to the Double type, and must keep one digit after the decimal point. For example, `cast("1" as double) = 1.0`.
- Explicit conversions of unsupported types may return an exception.
- If a conversion fails during execution, the conversion is aborted with an exception.
- To convert the Datetime type, use the default format yyyy-mm-dd hh:mi:ss. For more information, see *Conversions between the String type and the Datetime type*.

- Some types cannot be explicitly converted, but can be converted using built-in SQL functions. For example, the **to_char** function can be used to convert values of the Boolean type to the String type. For more information, see *TO_CHAR*. The **to_date** function can be used to convert values of the String type to the Datetime type. For more information, see *TO_DATE*.
- For more information, see *CAST*.
- If a DECIMAL value exceeds the value range, MSB overflow error or LSB overflow truncation may occur for CAST STRING TO DECIMAL.

Implicit conversion and scope

Implicit type conversion is an automatic type conversion performed by MaxCompute according to the usage context and type conversion rules. The following table lists the types that can be implicitly converted using MaxCompute.

	boole	tinyir	small	int	bigin	float	double	Decimal	string	varch	times	binary
boolean to	Y	N	N	N	N	N	N	N	N	N	N	N
tinyint to	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	N
smallint to	N	N	Y	Y	Y	Y	Y	Y	Y	Y	N	N
int to	N	N	Y	Y	Y	Y	Y	Y	Y	Y	N	–
bigint to	N	N	N	N	Y	Y	Y	Y	Y	Y	N	N
float to	N	N	N	N	Y	Y	Y	Y	Y	Y	N	–
double to	N	N	N	N	N	N	Y	Y	Y	Y	N	N
decimal to	N	N	N	N	N	N	N	Y	Y	Y	N	N
string to	N	N	N	N	N	N	Y	Y	Y	Y	N	N
varchar to	N	N	N	N	Y	Y	Y	Y	N	N	–	–
timestamp to	N	N	N	N	N	N	N	N	Y	Y	Y	N
binary to	N	N	N	N	N	N	N	N	N	N	N	Y

Y means can be converted. N means cannot be converted.

Note:

- The DECIMAL type and Datetime constant definition mode are added to MaxCompute2.0. 100BD indicates a DECIMAL, the value is 100. Datetime `2017-11-11 00:00:00` indicates a constant of the Datetime type. The constant definition is convenient because it can be directly used in values clauses and tables.
- In the earlier version of MaxCompute, values of the DOUBLE type can be implicitly converted to the BIGINT type. Owing to some reasons, such conversions may lead to data loss, which is not allowed by common database systems.

Common use:

```
select user_id+age+'12345',
           concat(user_name,user_id,age)
  from user;
```

Note:

- Implicit conversions of unsupported types may cause an error.
- If a conversion fails during execution, an exception occurs.
- MaxCompute automatically performs implicit conversions based on the context environment. We recommend that you use CAST to perform an explicit conversion when the types do not match.
- Implicit conversion rules are applicable to a specific range of scopes. In some scopes, only some rules can take effect. For more information, see the scopes of implicit conversions.

- **Implicit conversions under relational operators**

  Relational operators include equal to (=), not equal to (<>), less than (<), less than or equal to (<=), greater than (>), greater than or equal to (>=), IS NULL, IS NOT NULL, LIKE, RLIKE, and IN. For the particularities, implicit conversion rules of LIKE, RLIKE, and IN are discussed separately. The following descriptions do not contain these three special operators.

  The following table describes implicit conversion rules when different types of data is involved in relational operations.

From/To	Bigint	Double	String	Datetime	Boolean	Decimal
Bigint	–	Double	Double	N	N	Decimal
Double	Double	–	Double	N	N	Decimal
String	Double	Double	–	Datetime	N	Decimal

From/To	Bigint	Double	String	Datetime	Boolean	Decimal
Datetime	N	N	Datetime	–	N	N
Boolean	N	N	N	N	–	N
Decimal	Decimal	Decimal	Decimal	N	N	-

Note:

- If two types cannot be implicitly converted, the relational operation is aborted by an error.
- For more information about the relational operators, see *Relational Operators*.

- **Implicit conversions under special relational operators**

  Special relational operators include LIKE, RLIKE, and IN.

  - The usage of LIKE and RLIKE is as follows:

    ```
    source like pattern;
    source rlike pattern;
    ```

    The following illustrates the notes for LIKE and RLIKE in implicit conversions:

    - The source and pattern parameters of LIKE and RLIKE can only be of the String type.
    - Other types can neither be involved in the operations nor be implicitly converted to the String type.

  - The usage of IN is as follows:

    ```
    key in (value1, value2, …)
    ```

    Implicit conversion rules of IN:

    - Data in the value column must be consistent.
    - To compare keys and values, if Bigint, Double, and String types are compared, convert them to Double type. If the Datetime and String types are compared, convert them to Datetime type. Conversions between other types are not allowed.

- **Implicit conversions under arithmetic operators**

  Arithmetic operators include addition (+), subtraction (-), multiplication (*), division (/), modulo (%), unary plus (+), and unary minus (-). Their implicit conversion rules are described as follows:

  - Only the String, Bigint, Double, and Decimal types can be involved in the operation.
  - The String type are implicitly converted to the Double type before the operation.

- When the Bigint and Double types are involved in the operation, the Bigint type is implicitly converted to the Double type.
- The Datetime and Boolean types are not allowed in the arithmetic operation.

- **Implicit conversions under logical operators**

  Logical operators include AND, OR, and NOT. Their implicit conversion rules are as follows:

  - Only the Boolean type can be involved in the logical operation.
  - Other types are not allowed in the logical operation, and cannot be implicitly converted to other types.

Implicit conversions for Built-in functions

MaxCompute SQL provides numerous system functions. You can calculate one or multiple columns of any row and output data of any type. Their implicit conversion rules are described as follows:

- To call a function, if the data type of an input parameter is different from that defined in the function, convert the data type of the input parameter to that defined in the function.
- Parameters of different built-in functions of MaxCompute SQL have different requirements on implicit conversions. For more information, see *Built-in Functions*.

Implicit conversions under CASE WHEN

For more information about CASE WHEN, see *CASE WHEN Expressions*. Its implicit conversion rules are listed as follows:

- If the types of the returned values are Bigint and Double, convert all to the Double type.
- If a String type exists in return types, convert all to the String type. If the conversion fails (such as Boolean type conversion), an error is returned.
- Conversions between other types are not allowed.

Conversions between the String Type and Datetime Type

MaxCompute supports conversions between the String type and Datetime type. The conversion format is `yyyy-mm-dd hh:mi:ss`.

Unit	String (case-insensitive)	Value range
Year	yyyy	0001 - 9999
Month	mm	01 - 12
Day	dd	01 - 28,29,30,31

Unit	String (case-insensitive)	Value range
Hour	hh	00 - 23
Minute	mi	00 - 59
Second	ss	00 - 59

Note:

- In the value range of each unit, if the first digit is 0, it cannot be ignored. For example, `2014-1-9 12:12:12` is an invalid Datetime format and it cannot be converted from the STRING type to the Datetime type. It must be written as `2014-01-09 12:12:12`.
- Only the String type that meets the preceding format requirements can be converted to the Datetime type. For example, `cast("2013-12-31 02:34:34" as datetime)` converts`2013-12-31 02:34:34` of the String type to the Datetime type. Similarly, when the Datetime type is converted to the String type, the default conversion format is yyyy-mm-dd hh:mi:ss.

For example, the following conversions return an exception:

```
cast("2013/12/31 02/34/34" as datetime)
cast("20131231023434" as datetime)
cast("2013-12-31 2:34:34" as datetime)
```

The threshold of dd depends on the actual days of a month. If the value exceeds the actual days of the month, the conversion is aborted with an error.

Example:

```
cast("2013-02-29 12:12:12" as datetime) -- Returns an error because
February 29, 2013 does not exist.
cast("2013-11-31 12:12:12" as datetime) -- Returns an exception
because November 31, 2013 does not exist.
```

MaxCompute provides the TO_DATE function to convert the String type that does not meet the Datetime format to the Datetime type. For more information, see *TO_DATE*.

4.4 DDL SQL

4.4.1 Table Operations

This article shows you how to create, view, delete, rename and modify table information through the client.

Create tables

Statement format:

```
CREATE [EXTERNAL] 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], ...)]
[STORED BY StorageHandler] -- Limited to external tables
[WITH SERDEPROPERTIES (Options)] -- Limited to external tables
[LOCATION OSSLocation];-- Limited to external tables
[LIFECYCLE days]
[As select_statement]
 CREATE TABLE [IF NOT EXISTS] table_name
 LIKE existing_table_name
```

Consider the following points:

- When a table is created, an error is returned if the same name table exists without specifying the "if not exists" option. If the option is specified, no matter whether a same name table exists and even if the source table structure and the target table structure are inconsistent, all return successfully. The Meta information of the existing table does not change.
- Both the table name and column name are case insensitive and cannot have special characters. It must begin with a letter and can include a-z, A-Z, digits, and underscores (_). The name length cannot exceed 128 bytes.
- 1200 column definitions are allowed in a table.
- The data types support Bigint、Double、Boolean、Datetime、Decimal and String, MaxCompute2.0 extends many *data types*.

  Note:

  Once data type such asTinyint、Smallint、 Int、 Float、Varchar or TIMESTAMP BINARY is involved when running an SQL statement, `set odps.sql.type.system.odps2=true;` must be added before the SQL statement. The set statement and SQL statement are submitted simultaneously.
- Use Partitioned by to specify the *partition* and now Tinyint、Smallint、 Int、 Bigint、Varchar and String are supported.

The partition value cannot have a double byte characters (for example, Chinese), and must begin with an uppercase or a lowercase letter, followed by letter or a number. The name length cannot exceed 128 bytes. Special characters can be used, which include space, colon (:), underscore (_), dollar sign ($), pound sign (#), period (.), exclamation point (!), and '@'. Other characters such as (\t), (\n), (/), and so on are considered as undefined characters. When using partition fields in a partition table, to improve the processing efficiency, a full table scan is not needed to add, update, and read the data in a partition.

- Currently, 60,000 partitions are allowed in a table, and the partition hierarchy cannot exceed 6 levels.
- The comment content is the effective string and its length must not exceed 1024 bytes
- Lifecycle indicates the lifecycle of the table, the unit is 'days'. The statement create table like does not copy the lifecycle attribute from source table
- For more information about external tables, see *Access OSS*.

For example:

Assume that the table sale_detail is created to store sale records. The table uses sale_date and region as partition columns. Table creation statements are described as follows:

```
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 sale_detail.
```

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

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

If the table sale_detail has data, the example mentioned preceding copies all data of sale_detail into the table sale_detail_ctas1.

Note:

sale_detail is a partitioned table, while the table created by the statement `create table...as select...` does not copy the partition attribute. The partition column of source table becomes a general column of object table. In other words, sale_detail_ctas1 is a non-partition table with 5 columns.

In the statement `create table...as select...` if using a constant as a column value in Select clause, it is suggested specify the column name, such as:

```
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 statement is as shown as follows:

```
create table sale_detail_ctas3 as
        select shop_name,
            customer_id,
            total_price,
            '2013',
            'China'
        from sale_detail;
```

Then the forth column and fifth column of the created table sale_detail_ctas3 become system generated names, like `_c3`, `_c4`.

To let the destination table have the same structure as the source table, try to use `create table … like'` statement, such as:

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

Now the table structure of `sale_detail_like` is exactly the same as `sale_detail`. Except the life cycle, attributes including the column name, column comment, and table comment, of the two tables are the same. But the data in `sale_detail` cannot be copied into the table `sale_detail_like`.

View table information

Statement format:

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

For example:

- To view the info of the preceding table sale_detail, run the following statement:

  ```
  desc sale_detail;
  ```

  Return info:

  ```
  odps@ $odps_project>desc sale_detail;
  +------------------------------------------------------------------------
  +
  ```

```
| Owner: ALIYUN$lili.ll@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 infomation of the preceding table sale_detail_like, run the following statement:

```
desc sale_detail_like
```

Return info:

```
odps@ $odps_project>desc sale_detail_like;
+------------------------------------------------------------------------
+
| Owner: ALIYUN$lili.ll@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 preceding example, we can see that the attributes of sale_detail_like coincide with that of sale_detail, except for the lifecycle. For more information, see *Describe Table*.

Check the information of sale_detail_ctas1, you can find that sale_date and region are only normal columns and not partitions of the table.

Drop a table

Statement format:

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

Note:

- If the option [if exists] is not specified and the table does not exist, exception returns. If this option is specified, no matter whether the table exists or not, all return success.
- Data in OSS is not deleted when the external tables are deleted.

For example:

```
create table sale_detail_drop like sale_detail;
    drop table sale_detail_drop;
    --If the table exists, return success; otherwise, return exception
.
    drop table if exists sale_detail_drop2;
    --No matter whether the table sale_detail_drop2 exists or not, all
 return success.
```

Rename a table

Statement format:

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

Note:

- Rename operation is used to update the table name only and not the data in the table.
- If the new_table_name is duplicated an error may occur.
- If the table table_name does not exist, error may occur.

For example:

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

Alter Table Comments

Command format:

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

Note:

- The table table_name must exists.
- The comment length must not exceed 1024 bytes.

For example:

```
alter table sale_detail set comment 'new comments for table sale_detai
l';
```

Use the command **desc** to view the comment modification in the table. For more information, see *Describe Table*.

Alter Table LastDataModifiedTime

MaxCompute SQL supports **touch** operation to modify LastDataModifiedTime of a table. The result is to modify LastDataModifiedTime of a table to be current time.

Statement format:

```
ALTER TABLE table_name TOUCH;
```

Note:

- If the table table_name does not exist, an error is returned.
- This operation changes the value of LastDataModifiedTime of a table and this is when MaxCompute identifies change in the table data and then begins the corresponding lifecycle calculation.

Empty data from a non-partitioned table

Empty the data in specified non-partition table, This command does not support partition table. For the partition table, use `ALTER TABLE table_name DROP PARTITION` to clear the data in partition.

Command format:

```
TRUNCATE TABLE table_name;
```

4.4.2 Lifecycle of table

MaxCompute provides data life cycle management functions to facilitate you to release storage space and simplify the process of data recovery.

Modify lifecycle of table

Statement format:

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

Note:

- The parameter ‘days’ refers to the time required to complete the lifecycle. It must be a positive integer and its unit is ‘day’.
- Suppose that the table ‘table_name’ is a no-partition table. Calculated from the last updated date, the data is still not modified after N (days) days, then MaxCompute automatically recycles the table without user intervention (similar to ‘drop table’ operation).

- In MaxCompute, once the data in the table is modified, the LastDataModifiedTime is updated . So MaxCompute judges whether to recycle this table based on the LastDataModifiedTime setting and lifecycle.
- · Suppose the table ‘table_name’ is a partition table. MaxCompute determines whether to recycle the table according to LastDataModifiedTime of each partition.
- · Unlike no-partition table, after the last partition of a partitioned table has been recycled, the table is not deleted.
- · The lifecycle can be set for a table, not for the partition.
- · It can be specified while creating a table.

Example:

```
create table test_lifecycle(key string) lifecycle 100;
 -- Create a new table test_lifecycle and the lifecycle is 100 days.
 alter table test_lifecycle set lifecycle 50;
 -- Alter the lifecycle for the table test_lifecycle and set it to be
50 days.
```

Disable lifecycle of table

In some cases, the data in specified partitions do not need to be recycled by the lifecycle function . For example, data in the beginning of the month, or the data during the Global Shopping Day period. You can disable the lifecycle function using some specific partitions.

Statement format:

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

An example is shown as follows.

```
ALTER TABLE trans PARTITION(dt='20141111') DISABLE LIFECYCLE;
```

4.4.3 Column and Partition operation

This article shows you how to add, delete, and modify table partition command operations.

Add partition

Statement format:

```
ALTER TABLE TABLE_NAME ADD [IF NOT EXISTS] PARTITION partition_spec
partition_spec:(partition_col1 = partition_col_value1, partition_col2
 = partiton_col_value2, ...)
```

Note:

- Only ‘creating partitions’ are supported wherein, ‘creating partition columns’ are not supported.

- If the same name partition has already existed and the option [if not exists] is not specified, an exception returns.
- Currently, the maximum number of partitions supported in a single MaxCompute table is 60, 000.
- For tables that have multi-level partitions, to add a new partition, all partition values must be specified.

Example:

add a new partition for the table ‘sale_detail’.

```
alter table sale_detail add if not exists partition (sale_date='201312
', region='hangzhou');
-- Add partition successfully, to store the sale detail of hangzhou
region in December of 2013.
alter table sale_detail add if not exists partition (sale_date='201312
', region='shanghai');
-- Add partition successfully, to store the sale detail of shanghai
region in December of 2013.
alter table sale_detail add if not exists partition(sale_date='
20111011');
-- Only specify a partition sale_date, error occurs and return.
alter table sale_detail add if not exists partition(region='shanghai
');
-- Only specify a partition region, error occurs and return.
```

Drop partition

Delete the syntax format for the partition is as follows:

```
ALTER TABLE TABLE_NAME DROP [IF EXISTS] PARTITION partition_spec;
partition_spec:(partition_col1 = partition_col_value1, partition_col2
 = partiton_col_value2, ...)
```

Note:

If the partition does not exist and the option [if exists] is not specified, then an error returns.

Example:

delete a partition from the table sale_detail.

```
alter table sale_detail drop if exists partition(sale_date='201312',
region='hangzhou');
```

```
-- -Delete the sale details of Hangzhou in December of 2013 successful
ly.
```

Add column

Statement format:

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

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

Note:

You cannot specify order for a new column. By default, a new column is placed in the last column.

Modify column name

Statement format:

```
ALTER TABLE table_name CHANGE COLUMN old_col_name RENAME TO new_col_na
me;
```

Note:

- Column ‘old_col_name’ refers to an existing column.
- A column named ‘new_col_name’ cannot exist in the table.

Alter Column/Partition Comment

Modify column/partition comment is as follows:

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

Note:

The maximum comment content is 1024 bytes.

Modify column names and column notes simultaneously

Statement format:

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

Note:

- Column ‘old_col_name’ must be an existing column.

- A column named ‘new_col_name’ cannot exist in the table.
- The content of the comment cannot exceed 1024 bytes.

Modify LastDataModifiedTime of table/partition

MaxCompute MaxCompute SQL supports ‘touch’ operation to modify LastDataModifiedTime of a partition. The result is to modify ‘LastDataModifiedTime’ of a partition to be current time.

Statement format:

```
ALTER TABLE table_name TOUCH PARTITION(partition_col='partition_
col_value', ...)
```

Note:

- If ‘table_name’ or ‘partition_col’ does not exist, an error returns.
- If the specified partition_col_value does not exist, an error returns.
- This operation changes the value of ‘LastDataModifiedTime’ in the table and now MaxCompute determines whether the data of the table or partition has changed and the lifecycle calculation begins again.

Modify partition value

MaxCompute SQL supports to change the partition value for corresponding partition value through ‘rename’ operation.

Statement format:

```
ALTER TABLE table_name PARTITION (partition_col1 = partition_
col_value1, partition_col2 = partiton_col_value2, ...)
RENAME TO PARTITION (partition_col1 = partition_col_newvalue1,
partition_col2 = partiton_col_newvalue2, ...)
```

Note:

- The name of a partition column cannot be modified. Only the values in that column can be altered.
- To modify values in one or more partitions among multi-level partitions, users must write values for partitions at each level.

4.4.4 View operations

Create view

Statement format:

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

Note:

- To create a view, you must have 'read' privilege on the table referenced by view.
- Views can only contain one valid 'select' statement.
- Other views can be referenced by a view, but this view cannot reference itself. Circular reference is not supported.
- Writing the data into a view is not allowed, such as, using **`insert into`** or **`insert overwrite`** to operate view
- After a view was created,it may be inaccessable if the referenced table is altered, such as deleting a referenced table. You must maintain corresponding relationship between referenced tables and views.
- If the option 'if not exists' is not specified and the view has already existed, using **`create view`** causes abnormality. If this situation occurs, use **`create or replace view`** to recreate a view. After reconstruction, the privileges keep unchanged.

Example:

```
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;
```

Drop view

Statement format:

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

Note:

If the view does not exist and the option [if exists] is not specified, error occurs.

Example:

```
DROP VIEW IF EXISTS sale_detail_view;
```

Rename view

Statement format:

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

Note:

If the same name view has already existed, error occurs.

Example:

```
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;
    alter view sale_detail_view rename to market;
```

4.5 Insert Operation

4.5.1 INSERT OVERWRITE/INTO

Function definition:

```
INSERT OVERWRITE|INTO TABLE tablename [PARTITION (partcol1=val1,
partcol2=val2 ...)] [(col1,col2 ...)]
select_statement
FROM from_statement;
```

Note:

- Insert syntax of MaxCompute is different from MySQL or Oracle Insert syntax. The keyword table must be added following **`insert overwrite|into`**, instead of using tablename directly.
- When the target table for Insert is a partitioned table, expressions such as functions are not allowed in `[PARTITION (partcol1=val1, partcol2=val2 …)]`.
- Currently, INSERT OVERWRITE does not support inserting columns. You can use INSERT INTO instead.

Insert overwrite/into saves calculation results into a destination table.

The difference between **insert into** and **insert overwrite** is that **insert into** inserts added data into the table or partition, while **insert overwrite** clears source data from the table or partition before inserting the data in it.

Note:
The partition size in the MaxComputer partition table gets different data partition sizes when the same partition is repeatedly INSERT OVERWRITEd with the value described. This is because the file splitting logic changes when you select from the same partition on the same table and insert overwrite back to the same partition on the same table, thus causing the size of the data to change. But the total length of the data is constant around INSERT OVERWRITE, so users don't have to worry about billing for storage.

While processing data through MaxCompute SQL, **insert overwrite/into** is the most common statement. It can save the calculation result into a table, needed for the subsequent calculation. For example, use the following statements to calculate the sale detail of different regions from the table sale_detail:

```
create table sale_detail_insert like sale_detail;
alter table sale_detail_insert add partition(sale_date='2013', region
='china');
insert overwrite table sale_detail_insert partition (sale_date='2013
', region='china')
select shop_name, customer_id, total_price from sale_detail;
```

Note:
The correspondence between source table and destination table depends on the column sequence in select clause, not the column name correspondence between the two tables. The following statement is still valid:

```
insert overwrite table sale_detail_insert partition (sale_date='2013
', region='china')
select customer_id, shop_name, total_price from sale_detail;
-- When the sale_detail_insert table is created, the column sequence
is as below:
-- shop_name string, customer_id string, total_price bigint
-- When data is inserted from sale_detail to sale_detail_insert, the
insertion sequence of sale_detail is as below:
-- customer_id, shop_name, total_price
-- Inserts data in sale_detail.customer_id into sale_detail_insert.
shop_name.
-- Inserts data in sale_detail.shop_name into sale_detail_insert.
customer_id.
```

To insert data into a partition, the partition column cannot appear in the Select list.

```
insert overwrite table sale_detail_insert partition (sale_date='2013
', region='china')
```

```
select shop_name, customer_id, total_price, sale_date, region from
sale_detail;
-- Returns an error. The items sale_date and region are partition
 columns, which cannot appear in the INSERT statement of static
partitions.
```

Simultaneously, the value of the partition can only be a constant and expressions cannot appear.

The following statements are invalid:

```
insert overwrite table sale_detail_insert partition (sale_date=
datepart('2016-09-18 01:10:00', 'yyyy') , region='china')
select shop_name, customer_id, total_price from sale_detail;
```

4.5.2 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 256 ways of output can be written in a single SQL statement. A syntax error occurs, if the output exceeds 256 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.
```

4.5.3 DYNAMIC PARTITION

To ‘insert overwrite’ into a partition table, specify the partition value in the statement. It can also be realized in a more flexible way, to specify a partition column in a partition table but not give the value.

OverView

Correspondingly, the columns in Select clause are used to specify these partition values.

Statement format:

```
insert overwrite table tablename partition (partcol1, partcol2 ...)
select_statement from from_statement;
```

Note:

- In the ‘select_statement’ field, the following field provides a dynamic partition value for the target table. If the target table has only one-level dynamic partition, the last field value of select_statement is the dynamic partition value of the target table.
- Currently, a single worker can only output up to 512 dynamic partitions in a distributed environment, otherwise it leads to abnormality.
- Currently, any dynamic partition SQL cannot generate more than 2,000 dynamic partitions; otherwise it causes abnormality.
- The value of dynamic partition cannot be NULL, and also does not support special or Chinese characters, otherwise an exception is thrown. The exception is as follows:

```
FAILED: ODPS-0123031:Partition exception - invalid dynamic
partition value:
```

```
                province=xxx
```

- If the destination table has multi-level partitions, it is allowed to specify parts of partitions to be static partitions through ‘Insert’ statement, but the static partitions must be advanced partitions .

Examples

A simple example to explain dynamic partition is as follows:

```
create table total_revenues (revenue bigint) partitioned by (region
string);
    insert overwrite table total_revenues partition(region)
        select total_price as revenue, region
            from sale_detail;
```

As mentioned in the preceding example, user is unable to know which partitions are generated before running SQL. Only after the Select statement running ends, user can confirm which partitions have been generated using ‘region’ as the value. This is why the partition is called as the **Dynamic Partition**.

Other Examples:

```
create table sale_detail_dypart like sale_detail; --Create target
table.
```

- --Example 1:

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

  - In ‘sales_detail’ table, the value of the sale_date determines the sales_date partition value of the target table, and the value of the region determines the region partition value of the target table.
  - In a dynamic partition, the correspondence between the select_statement field and the dynamic partition of the target table is determined by the order of the fields. In this example, if the Select statement is written as the following:

    ```
    select shop_name,customer_id,total_price,region,sale_date from
             sale_detail;
    ```

    the region value determines the sale_date partition value of the target table, and the value of sale_date determines the region partition value of the target table.

- --Example 2:

```
insert overwrite table sale_detail_dypart partition (sale_date='2013
', region)
        select shop_name,customer_id,total_price,region from
sale_detail;
    -- Return successfully; multiple partitions; specify a secondary
 partition.
```

- --Example 3:

```
insert overwrite table sale_detail_dypart partition (sale_date='2013
', region)
        select shop_name,customer_id,total_price from sale_detail;
    -- Return failure information. When inserting a dynamic
partition, the dynamic partition column must appear in Select list.
```

- --Example 4:

```
insert overwrite table sales partition (region='china', sale_date)
select shop_name,customer_id,total_price,region from sale_detail;
    -- Return failure information. User cannot specify the
lowsubpartition only, but needs to insert advanced partition
dynamically.
```

When the old version of MaxCompute performs dynamic partitioning, if the partition column type is not exactly the same as the column type in the corresponding select list, an error is reported. MaxCompute 2.0 supports implicit conversion, as shown in the following :

```
create table parttable(a int, b double) partitioned by (p string);
insert into parttable partition(p) select key, value, current_ti
mestmap() from src;
select * from parttable;
```

The result is as follows:

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

4.5.4 VALUES

This article introduces you to `INSERT … VALUES` command operation.

In the test phase, prepare some basic data for a small data table. You can quickly write some test data to the test table by using the `INSERT … VALUES` statement.

Note:

Currently, INSERT OVERWRITE does not support insert columns, use `INSERT INTO` instead.

Statement format:

```
INSERT INTO TABLE tablename
[PARTITION (partcol1=val1, partcol2=val2 ...)][co1name1,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 run successfully, the result of partition ‘abc’ is as follows:

```
| key | value | p |

| a | 1 | abc |
| b | 2 | abc |
| c | 3 | abc |
```

When many columns are in the table, and you want to insert data into some of the columns , 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 run successfully, 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 ‘insert into…select…’.

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) are to define a table named t whose columns are a and b, data type is (a string, b bigint), the data type of which is 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 run successfully, 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 union all to spell out the constants. As follows:

```
select 1 c from dual
union all
select 2 c from dual;
--等同于
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, select can be run without the from statement, if the expression list of select 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 UDF, etc., 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. 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 datetime or timestamp type through values, specify the type name in 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');
```

4.6 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. The lateral view first calls UDTF for each row of the original table. UTDF splits a row into one or more rows. The lateral view combines the results to produce a virtual table that supports the alias table.

Single Lateral View statement

Syntax:

```
lateralView: LATERAL VIEW [OUTER] udtf(expression) tableAlias AS
columnAlias (',' columnAlias) * fromClause: FROM baseTable (lateralVie
w)*
```

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”

4.7 Select Operation

4.7.1 Introduction to the SELECT Syntax

This article introduces you to the Select grammar format and the precautions for executing nested queries, sorting operations and grouping queries using select grammar.

Introduction to the SELECT Syntax

The command 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]
```

Note:

- When using SELECT to read data from the table, specify the names of the columns to be read, or use an asterisk (*) to represent all columns. A simple SELECT statement is shown as follows:

  ```
  select * from sale_detail;
  ```

  To read only the shop_name column in sale_detail, use the following statement:

  ```
  select shop_name from sale_detail;
  ```

  Use where to specify filtering conditions. For example:

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

  When a Select statement is used, a maximum of 10,000 rows of results can be displayed. But if the Select statement serves as a clause, all the results are returned to the upper-level query.
- Full table scan is prohibited when you select a partitioned table.

  For new projects created after January 10, 2018, 20:00 (UTC+8) full table scan is not allowed for the partitioned table in the project by default When SQL runs. Partitions to be scanned must be specified in partition conditions to reduce unnecessary SQL I/O, and computing resources, and the unnecessary cost. Note: Using the Pay-As-You-Go billing method, the amount of data input is one of the billing parameters.

If the table definition is `t1(c1,c2) partitioned by(ds)`, running the following statement in a new project is restricted and 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 scans the entire table of the partitioned table. The set statement must be submitted along with the SQL statement. Suppose 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';
```

- The filter conditions supported by 'where' clause are shown as follows:

Filter conditions	Description
> 、 < 、 =、 >=、 <=、 <>	Relational operators
like、rlike	The source and pattern parameters of like and rlike can only be of the String type.
in、not in	If a subquery is attached to the in or not in 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 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 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';
```

- **`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 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.
  ```

- **`group by`**: Query by group. It is generally used together with an aggregate function. A Select statement that contains an aggregate function follows these rules:

  1. The key using group by can be the name of a column in the input table.
  2. Alternatively, it can be an expression consisting of columns of the input table. The key cannot be the alias of an output column of the Select statement.
  3. Rule i takes precedence over rule ii. If rules i and ii conflict, that is, if the key using group by is a column or expression of the input table and an output column of Select, rule i prevails.

  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
  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_pric
  e;
  -- 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
```

These 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*.

- 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 a maximum of 10,000 rows. The upper limit of screen display varies with projects, which can be controlled through the **`setproject`** console.

- 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;
  ```

- 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.
```

- order by or group by cannot be used together with distribute by/sort] by. Aliases of SELECT output columns must be used.

Note:

- The keys of order by/sort by/distribute by must be output columns (namely, column aliases) of Select statements.
- In MaxCompute SQL parsing, order by/sort by/distribute by come after Select operations. Therefore, they can only accept the output columns of Select statements as keys.

4.7.2 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`.

4.7.3 Subquery

Basic definition of a subquery

A normal SELECT operation reads data from several tables, for example, select `column_1 , column_2 … from table_name`. However, the query object can be another SELECT operation, which is shown as follows:

```
select * from (select shop_name from sale_detail) a;
```

Note:

The subquery must have an alias.

In a FROM clause, a subquery can be used as a table to perform JOIN operations with other tables or subqueries, which is shown as follows:

```
create table shop as select * from sale_detail;
select a.shop_name, a.customer_id, a.total_price from
(select * from shop) a join sale_detail on a.shop_name = sale_detail.
shop_name;
```

IN SUBQUERY / NOT IN SUBQUERY

IN SUBQUERY is similar to LEFT SEMI JOIN.

For example:

```
SELECT * from mytable1 where id in (select id from mytable2);
-- is equivalent to
SELECT * from mytable1 a LEFT SEMI JOIN mytable2 b on a.id=b.id;
```

Currently, MaxCompute supports both IN SUBQUERY and CORRELATED conditions.

For example:

```
SELECT * from mytable1 where id in (select id from mytable2 where
value = mytable1.value);
```

`where value = mytable1.value` in the subquery is a CORRELATED condition. MaxCompute of early versions reports errors for such expressions that reference source tables both in subqueries and in outer queries. **MaxCompute supports such expressions now**. In fact, such filtering conditions are a part of the ON condition in SEMI JOIN.

NOT IN SUBQUERY is similar to LEFT ANTI JOIN. However, they have one significant difference .

For example:

```
SELECT * from mytable1 where id not in (select id from mytable2);
-- If none of the IDs in mytable2 are NULL, this statement is
equivalent to
SELECT * from mytable1 a LEFT ANTI JOIN mytable2 b on a.id=b.id;
```

If mytable2 contains any column whose ID is NULL, the NOT IN expression is NULL, so that the WHERE condition is invalid and no data is returned. This is different from LEFT ANTI JOIN.

MaxCompute 1.0 supports [NOT] IN SUBQUERY not serving as a JOIN condition, for example , in a non-WHERE statement, or failure in conversion to a JOIN condition even in a WHERE statement. MaxCompute 2.0 still supports this feature. However, [NOT] IN SUBQUERY cannot be converted to SEMI JOIN, and a separate job must be started to run subqueries. Therefore, [NOT] IN SUBQUERY does not support CORRELATED conditions.

For example:

```
SELECT * from mytable1 where id in (select id from mytable2) OR value
 > 0;
```

As the WHERE clause includes OR, [NOT] IN SUBQUERY cannot be converted to SEMI JOIN. A separate job must be started to run subqueries.

In addition, partition tables are specially processed:

```
SELECT * from sales_detail where ds in (select dt from sales_date);
```

If ds is a partition column, `select dt from sales_date` separately starts a job to run subqueries, instead of converting to SEMI JOIN. After running, the results are compared with ds one by one. If a ds value in sales_detail is not in the returned results, the partition is not read to make sure that partition pruning is still valid.

EXISTS SUBQUERY/NOT EXISTS SUBQUERY

In an EXISTS SUBQUERY, when at least one data row exists in the subquery, TRUE is returned; otherwise, FALSE is returned. NOT EXISTS subquery is completely opposite of this.

Currently, MaxCompute supports only subqueries including the correlated WHERE conditions. EXISTS SUBQUERY/NOT EXISTS SUBQUERY is implemented by converting to LEFT SEMI JOIN or LEFT ANTI JOIN.

For example:

```
SELECT * from mytable1 where exists (select * from mytable2 where id
 = mytable1.id);
-- is equivalent to
Select * From mytable1 a left semi join mytable2 B on A. ID = B. ID;
```

While

```
SELECT * from mytable1 where not exists (select * from mytable2 where
id = mytable1.id);
-- is equivalent to
SELECT * from mytable1 a LEFT ANTI JOIN mytable2 b on a.id=b.id;
```

4.7.4 UNION ALL/UNION [DISTINCT]

The UNION operator is used to merge the result set of two or more SELECT statements.

The syntax format is as follows:

```
select_statement UNION ALL select_statement;
select_statement UNION [DISTINCT] select_statement;
```

- **`UNION ALL`**: Combines two or multiple data sets returned by a SELECT operation into one data set. If the result contains duplicated rows, all rows that meet the conditions are returned, and deduplication of duplicated rows is not applied.
- **`UNION [DISTINCT]`**: In this statement, DISTINCT can be ignored. It combines two or multiple data sets returned by a SELECT operation into one data set. If the result contains duplicated rows, deduplication is applied.

Following is an example of the UNION ALL operation:

```
Select * From sale_detail where region = 'Hangzhou'
       union all
select * from sale_detail where region = 'shanghai';
```

Following is an example of the UNION operation:

```
SELECT * FROM src1 UNION SELECT * FROM src2;
--The execution effect is equivalent to
```

```
SELECT DISTINCT * FROM (SELECT * FROM src1 UNION ALL SELECT * FROM
src2) t;
```

Note:

- The number and types of queried columns corresponding to the UNION ALL/UNION operation must be consistent.
- Generally, MaxCompute allows UNION ALL/UNION operations performed on a maximum of 256 tables. A syntax error is returned if the number of tables exceeds this limit.

The meaning of LIMIT following UNION:

If UNION is followed by CLUSTER BY, DISTRIBUTE BY, SORT BY, ORDER BY, or a LIMIT clause, the clause has an effect on all the preceding UNION results rather than the last SELECT statement of UNION. Currently, MaxCompute adopts this action in `set odps.sql.type.system.odps2=true;`.

For 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) ORDER BY a LIMIT 3;
```

The returned result is as follows:

```
| a |

| 0 |
| 1 |
| 2 |
```

4.7.5 JOIN operation

The JOIN operation of MaxCompute supports n-way join, but does not support Cartesian product, that is, a link without the ON condition.

Function definition.

```
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 is an equality expression.

LEFT OUTER JOIN: Returns all records from the left table (shop) even if no matching row exists in the right table (sale_detail).

```
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;
    -- As the tables shop and sale_detail both have the shop_name
column, aliases must be used in the select clause for distinguishing.
```

RIGHT OUTER JOIN: indicates the right join. It returns all records from the right table even if no matching record exists in the left table.

For example.

```
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: indicates the full join. It returns all records from both the left and the right table.

For example.

```
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;
```

If at least one matching record exists in the table, INNER JOIN returns the row. The keyword INNER can be ignored.

```
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;
```

The join condition only allows equivalent conditions connected using and. Only MAPJOIN supports non-equivalent join conditions or multiple conditions connected using or.

```
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;
     -- Supports n-way JOIN examples
select a.* from shop a join sale_detail b on a.shop_name ! = b.
shop_name;
```

```
    -- Returns an error because non-equivalent JOIN conditions are not
 supported
```

IMPLICIT JOIN, MaxCompute supports the following JOIN method:

```
SELECT * FROM table1, table2 WHERE table1.id = table2.id;
--The execution effect is equivalent to
SELECT * FROM table1 JOIN table2 ON table1.id = table2.id;
```

4.7.6 SEMI JOIN

MaxCompute supports SEMI 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. Supported syntaxes include: LEFT SEMI JOIN and LEFT ANTI JOIN.

LEFT SEMI JOIN

When a JOIN condition is valid, data in the left table is returned. That is, if the ID of a row in mytable1 appears in all IDs in mytable2, this row is saved in the result set.

For example:

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

Only the data in mytable1 is returned if the ID of mytable1 appears in the ID of mytable2.

LEFT ANTI JOIN

When a JOIN condition is invalid, data in the left table is returned. That is, if the ID of a row in mytable1 does not appear in any ID in mytable2, this row is stored in the result set.

For example:

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

Only the data in mytable1 is returned if the ID of mytable1 does not appear in the ID of mytable2.

4.7.7 MAPJOIN HINT

MapJoin helps to join a large table with one or multiple small tables.It is faster than common Join operations.

A typical scenario of MapJoin, is as follows: When the data volume is small, SQL loads all your specified small tables into the memory of the program performing the Join operation to speed up JOIN execution.

Note:

When you use the MapJoin, note the following:

- The left table of ‘left outer join’ must be a big table.
- The right table of right outer join must be a big table.
- For INNER JOIN, both the left and right tables can be large tables.
- For FULL OUTER JOIN, MapJoin cannot be used.
- MapJoin supports small tables as subqueries.
- When MapJoin is used and a small table or subquery must be referenced, the alias must be referenced.
- MapJoin supports non-equivalent JOIN conditions or multiple conditions connected using OR.
- Currently, MaxCompute allows a maximum of eight small tables to be specified in MapJoin. Otherwise, a syntax error is returned.
- If MapJoin is used, the total memory occupied by all small tables cannot exceed 512 MB. Note that MaxCompute uses compressed storage, so the data size is sharply expanded after small tables are loaded into the memory. The limit of 512 MB refers to the size after small tables are loaded into the memory.
- When JOIN is performed on the multiple tables, the two leftmost tables cannot be tables for MapJoin at the same time.

For 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;
```

MaxCompute SQL does not support complex JOIN conditions, such as non-equivalent expressions and the OR logic, in the ON condition of common JOIN operations. However, MapJoin supports such operations.

For example:

```
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;
```

4.7.8 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
```

4.7.9 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.

4.7.10 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.

4.8 Builtin functions

4.8.1 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:

```
 if trans_date = 2005-02-28 00:00:00
dateadd(trans_date, 1, 'dd') = 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.
dateadd(trans_date, -1, 'dd') = 2005-02-27 00:00:00
-- Minus one day.
dateadd(trans_date, 20, 'mm') = 2006-10-28 00:00:00
-- Add 20 months. The month spillover is caused and the year is added
 ‘1’.
If trans_date = 2005-02-28 00:00:00, dateadd(transdate, 1, 'mm') =
2005-03-28 00:00:00
If trans_date = 2005-01-29 00:00:00, dateadd(transdate, 1, 'mm') =
2005-02-28 00:00:00
-- No 29th is in Feb. of 2005. The date is intercepted to the last day
 of current month.
 If trans_date = 2005-03-30 00:00:00, dateadd(transdate, -1, 'mm') =
2005-02-28 00:00:00
```

Note:

Here the value of trans_date used only as an example. This simple expression is often used to present the datetime in this file.

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 datatime 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:

- **datet1**, **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 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('2013-05-31 13:00:00', '2013-05-31 12:30:00', 'ss') =
1800
    datediff('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('2013-06-08 01:10:00', 'yyyy') = 2013
datepart('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 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('2011-12-07 16:28:46', 'yyyy') = 2011-01-01 00:00:00
datetrunc('2011-12-07 16:28:46', 'month') = 2011-12-01 00:00:00
datetrunc('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 correspond ing 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 datatime 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 expecion is indicated. If redundant format strings appear in **`format`**, then get the datatime value corresponding to the first format string, other strings are taken as separators. For example, `to_date (‘1234-2234’, ‘yyyy-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', 'Alibabayyyy-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', 'Alibabayyyy-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('2010-12-03 00:00:00', 'Alibabayyyy-mm*dd') = 'Alibaba2010-12*
03'
to_char('2008-07-18 00:00:00', 'yyyymmdd') = '20080718'
to_char('Alibaba2010-12*3', 'Alibabayyyy-mm*dd') -- Format is not
compatible and exception is thrown.
```

```
to_char('2010-24-01', 'yyyy') -- Format is not compatible and
exception is thrown.
to_char('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.

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.

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')=1970 from dual;
```

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) = null
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') = 3.9495967741
935485
months_between('1996-10-30','1997-02-28 10:30:00' ) = -3.9495967741
935485
months_between('1996-09-30','1996-12-31') = -3.0
```

4.8.2 Mathematical functions

This article introduces you to the mathematical function commands and instructions supported by MaxCompute SQL.

ABS

Function definition:

```
Double abs(Double number)
Bigint abs(Bigint number)
Decimal abs(Decimal number)
```

Usage:

Returns an absolute value.

Parameter description:

number: It is any number of Type Double, Bigint, or Decimal.

- If the input is Bigint and return Bigint.
- If the input is Double, return Double.
- If the input is Decimal, return Decimal.

If the input is String, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

The return result depends on the type of input parameter. Example, if the input is null, return null.

Note:

When the value of input Bigint type exceeds the maximum value of Bigint, return Double type. In this case, the precision may be absent.

Example:

```
abs(null) = null
abs(-1) = 1
abs(-1.2) = 1.2
abs("-2") = 2.0
abs(12232083745629837659238745692 3748) = 1.2232083745629837e32
```

The following is a completed ABS function example used in SQL. The use methods of other built-in functions (except Window Function and Aggregation Function) are similar.

```
select abs(id) from tbl1;
-- Take the absolute value of the id field in tbl1.
```

ACOS

Function definition:

```
Double acos(Double number)
Decimal acos(Decimal number)
```

Usage:

Calculates the inverse cosine of a number.

Parameter description:

number: Double or Decima type, -1<=number <=1. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type, the value is between 0 to π. If number is null, return null.

Example:

```
acos("0.87") = 0.5155940062460905
```

```
acos(0) = 1.5707963267948966
```

ASIN

Function definition:

```
Double asin(Double number)
Decimal asin(Decimal number)
```

Usage:

Calculates the inverse sine function of number.

Parameter description:

number: Double or Decima type, -1<=number <=1. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type, the value is between -π/2 to π/2. If the number is null, return null.

Example:

```
asin(1) = 1.5707963267948966
asin(-1) = -1.5707963267948966
```

ATAN

Function definition:

```
Double atan(Double number)
```

Usage:

Calculates the back-cut function of number.

Parameter description:

Number: Double type, if the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double type, the value is between -π/2 to π/2. If the number is null, return null.

Example:

```
atan(1) = 0.7853981633974483
```

```
atan(-1) = -0.7853981633974483
```

CEIL

Function definition:

```
Bigint ceil(Double value)
Bigint ceil(Decimal value)
```

Usage:

This function returns the smallest integral value not less than the argument.

Parameter description:

value: Double or Decimal type, If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Bigint type. If the number is null, return null.

Example:

```
ceil(1.1) = 2
ceil(-1.1) = -1
```

CONV

Function definition:

```
String conv(String input, Bigint from_base, Bigint to_base)
```

Usage:

Converts a number into a Hexadecimal number.

Parameter description:

- **input**: an integer to be converted, represented by String. Accept the implicit conversion of Bigint and Double.
- **from_base**, **to_base**: Decimal value, the acceptable values can be 2, 8, 10 and 16. Accept the implicit conversion of String and Double.

Return value:

Returns the String type. If the number is null, return null. The conversion process runs at a 64-bit precision. An exception is thrown when overflow occurs. If the input is a negative value (begin with

'-'), an exception is thrown. If the input value is a decimal, it is converted to an integer before hex conversion. The decimal part is excluded.

Example:

```
conv('1100', 2, 10) = '12'
conv('1100', 2, 16) = 'c'
conv('ab', 16, 10) = '171'
conv('ab', 16, 16) = 'ab'
```

COS

Function definition:

```
Double cos(Double number)
Decimal cos(Decimal number)
```

Usage:

Input is the radian value.

Parameter description:

number: Double or Decimal type. If the input is String, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type. If the number is NULL, return NULL.

Example:

```
cos(3.1415926/2)=2.6794896585028633e-8
cos(3.1415926)=-0.9999999999999986
```

COSH

Function definition:

```
Double cosh(Double number)
Decimal cosh(Decimal number)
```

Usage:

It is the Hyperbolic cosine function

Parameter description:

number: Double or Decimal type. If the input is String, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type. If the number is NULL, return NULL.

COT

Function definition:

```
Double cot(Double number)
Decimal cot(Decimal number)
```

Usage:

Inputs the radian value.

Parameter description:

number: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type. If the number is NULL, return NULL.

EXP

Function definition:

```
Double exp(Double number)
Decimal exp(Decimal number)
```

Usage:

It is the Exponential function.

Return value:

Returns the exponent value of number.

Parameter description:

number: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type. If the number is NULL, return NULL.

FLOOR

Function definition:

```
Bigint floor(Double number)
```

```
Bigint floor(Decimal number)
```

Usage:

Returns the largest integral value not greater than the argument.

Parameter description:

number: Double or Decimal type. If the input is String or Bigint type, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Bigint type. If the input is null, return null.

Example:

```
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

Function definition:

```
Double ln(Double number)
Decimal ln(Decimal number)
```

Usage:

Returns the natural logarithm of the number.

Parameter description:

number: Double or Decimal type.

- If the input is String or Bigint type, it is converted to Double by implicit conversion. If the input is another type, an error occurs.
- If the number is null, return null. If number is negative or 0, an exception is thrown.

Return value:

Returns the Double or Decimal type.

LOG

Function definition:

```
Double log(Double base, Double x)
```

```
Decimal log (decimal base, decimal X)
```

Usage:

Returns the logarithm of x whose base number is base.

Parameter description:

- **base**: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.
- **x**: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the logarithm value of Double or Decimal type.

- If base or x is null, return null.
- If one of base or x is negative or zero, it causes abnormality.
- If base is 1, it also causes abnormality.

POW

Function definition:

```
Double pow(Double x, Double y)
Decimal pow(Decimal x, Decimal y)
```

Usage:

Return x to the yth power, that is x^y.

Parameter description:

- **X**: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.
- **Y**: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type. If X or Y is null, return null.

RAND

Function definition:

```
Double rand(Bigint seed)
```

Usage:

Return a random number (that changes from row to row), Specifying the seed makes sure the generated random number sequence is deterministic, Return value range is from 0 to 1.

Parameter description:

seed: Bigint type, random number seed, to determine starting values of the random number sequence.

Return Value:

Returns the Double type.

Example:

```
select rand() from dual;
select rand(1) from dual;
```

ROUND

Function definition:

```
Double round(Double number, [Bigint Decimal_places])
Decimal round(Decimal number, [Bigint Decimal_places])
```

Usage:

Four to five homes to the specific decimal point position.

Parameter description:

- **number**: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.
- **Decimal_place**: A Bigint type constant, four to five homes to the decimal point position. If it is other type, an exception is thrown. If you exclude it, it indicates four to five homes into a single digit. The default value is zero

Return value:

Returns the Double or Decimal type. If **number** or **Decimal_places** is null, return null.

Note:

Decimal_places can be negative. The negative is counted from decimal point to the left.

Deletethe decimal part. If decimal_place is greater than the length of the integer part, return 0.

Example:

```
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

Function definition:

```
Double sin(Double number)
Decimal sin(Decimal number)
```

Usage:

Calculates the sine function of number, the input is the radian value.

Parameter description:

number: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type. If the number is NULL, return NULL.

SINH

Function definition:

```
Double sinh(Double number)
Decimal sinh(Decimal number)
```

Usage:

Calculates the hyperbolic sine function of number.

Parameter description:

number: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type. If the number is NULL, return NULL.

SQRT

Function definition:

```
Double sqrt(Double number)
Decimal sqrt(Decimal number)
```

Usage:

Calculates the square root of number.

Parameter description:

number: Double or Decimal type, must be greater than zero, if it is less than zero, an exception occur. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type. If the number is NULL, return NULL.

TAN

Function definition:

```
Double tan(Double number)
Decimal tan(Decimal number)
```

Usage:

Calculates the tangent function of the number, the input is the radian value.

Parameter description:

number: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type. If the number is NULL, return NULL.

TANH

Function definition:

```
Double tanh(Double number)
Decimal tanh(Decimal number)
```

Usage:

Calculates the hyperbolic tangent function of number.

Parameter description:

number: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.

Return value:

Returns the Double or Decimal type. If the number is NULL, return NULL.

TRUNC

Function definition:

```
Double trunc(Double number[, Bigint Decimal_places])
Decimal trunc(Decimal number[, Bigint Decimal_places])
```

Usage:

This function is used to intercept the input number to a specified decimal point place.

Parameter description:

- **number**: Double or Decimal type. If the input is String or Bigint, it is converted to Double by implicit conversion. If the input is another type, an error occurs.
- **Decimal_places**: a Bigint type constant, the decimal point place to intercept the number. Other types are converted to Bigint. If this parameter is excluded, default to intercept to single digit.

Return value:

Returns the Double or Decimal type. If the **number** or **Decimal_places** is NULL, return NULL.

Note:

- If the Double type is returned, the display of the returned result may not be as expected, such as trunc(125.815, 1) (this problem exists in all the systems).
- The part to be truncated is supplemented by zero.
- Decimal_places can be negative. The negative is truncated from the decimal point to the left and delete the decimal part. If Decimal_place are greater than the length of the integer, return zero.

Example:

```
trunc(125.815) = 125.0
trunc(125.815, 0) =125.0
```

```
trunc(125.815, 1) = 125.80000000000001
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
```

Maxcomputerte2.0 New Extended Mathematical Functions

With the upgrade to MaxCompute 2.0, some mathematical functions have been added to the product. If a new function uses a new data type, add the following set statement before using the new functions SQL statement:

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

The new extended functions are described as follows.

LOG2

Function definition:

```
Double log2(Double number)
Double log2(Decimal number)
```

Note:

Before the SQL statement which uses the LOG2 function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

Returns the log base 2 of a specific number.

Parameter description:

`number`: Double or Decimal type.

Return Value:

Returns the Double type. If the input is zero or null, the returned value is null.

The example is as follows:

```
log2(null)=null
log2(0)=null
```

```
log2(8)=3.0
```

LOG10

Function definition:

```
Double log10(Double number)
Double log10(Decimal number)
```

Note:

Before the SQL statement which uses the LOG10 function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

Returns the log base 10 of the specific number.

Parameter description:

`number`: Double or Decimal type.

Return Value:

Returns the Double type. If the input is zero or null, the returned value is null.

The example is as follows:

```
log10(null)=null
log10(0)=null
log10(8)=0.9030899869919435
```

BIN

Function definition:

```
String bin(Bigint number)
```

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.

Usage:

Returns the binary code expression for the specific number.

Parameter description:

`number`: Bigint type.

Return value:

String type. If the input is zero, then zero is returned; if the input is null, null is returned.

Example:

```
bin(0)='0'
bin(null)='null'
bin(12)='1100'
```

HEX

Function definition:

```
String hex(Bigint number)
String hex(String number)
String hex (binary number)
```

Note:

Before the SQL statement which uses the HEX function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

This function is used to converts integers or characters to hexadecimal format.

Parameter description:

`number`: If number is of the Bigint type, the hexadecimal format of the number is returned. If this variable is a String type, the hexadecimal format of the string is returned.

Return value:

Returns the String type. If the input is zero, then zero is returned; if the input is null, an exception is returned.

Example:

```
hex(0)=0
hex('abc')='616263'
hex(17)='11'
hex('17')='3137'
hex(null) results in an exception and returns failed.
```

Note:

If the input parameter is a Binary type, add `set odps.sql.type.system.odps2=true;`, and submit it with SQL to use the new data type normally.

UNHEX

Function definition:

```
BINARY unhex(String number)
```

Note:

Before the SQL statement which uses the UNHEX function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

Returns the string represented by a given hexadecimal string.

Parameter description:

number: A hexadecimal string.

Return value:

Returns the Binary type. If the input is zero, failed is returned. If the input is null, null is returned.

Example:

```
Unhex ('616263') = 'abc'
unhex(616263)='abc'
```

RADIANS

Function definition:

```
Double radians(Double number)
```

Note:

Before the SQL statement which uses the RADIANS function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

This function is used to converts degrees to radians.

Parameter description:

number: Double type.

Return value:

Returns the Double type, if the input is null, null is returned.

Example:

```
radians(90)=1.5707963267948966
radians(0)=0.0
radians(null)=null
```

DEGREES

Function definition:

```
Double degrees(Double number)
Double degrees(Decimal number)
```

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.

Usage:

This function is used to converts radians to degrees.

Parameter description:

`number`: Double or Decimal type.

Return value:

Returns Double data type. If the input is null, null is returned.

Example:

```
degrees(1.5707963267948966)=90.0
degrees(0)=0.0
Degrees (null) = NULL
```

SIGN

Function definition:

```
Double sign(Double number)
Double sign(Decimal number)
```

Note:

Before the SQL statement which uses the SIGN function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

Applies the sign of the input data. 1.0 indicates a positive number and -1.0 indicates a negative number. Otherwise, 0.0 is returned.

Parameter description:

number: Double or Decimal type.

Return value:

Returns Double data type. If the input is 0, 0.0 is returned. If the input is null, null is returned.

Example:

```
sign(-2.5)=-1.0
Sign (2.5) = 1.0
sign(0)=0.0
sign(null)=null
```

E

Function definition:

```
Double e()
```

Note:

Before the SQL statement which uses the E function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

This function is used to return the e value.

Return Value:

Returns the Double type.

Example:

```
e()=2.718281828459045
```

PI

Function definition:

```
Double pi()
```

Note:

Before the SQL statement which uses the PI function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

This function is used to return the π value.

Return Value:

Returns the Double type.

Example:

```
pi()=3.141592653589793
```

FACTORIAL

Function definition:

```
Bigint factorial(Int number)
```

Note:
Before the SQL statement which uses the FACTORIAL function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

This function is used to return the factorial for the specific number.

Parameter description:

number: Int-type data, range: [0 –20].

Return value:

Returns the Bigint type, if the input is zero, one is returned. If the input is null or outside the range [0 –20], null is returned.

Example:

```
factorial(5)=120 --5! = 5*4*3*2*1 = 120
```

CBRT

Function definition:

```
Double cbrt(Double number)
```

Note:

Before the SQL statement which uses the CBRT function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

This function is used to return the cube root.

Parameter description:

`number`: Double type.

Return value:

Returns Double data type. If the input is null, null is returned.

Example:

```
cbrt(8)=2
cbrt(null)=null
```

SHIFTLEFT

Function definition:

```
Int shiftleft(Tinyint|Smallint|Int number1, Int number2)
Bigint shiftleft(Bigint number1, Int number2)
```

Note:

Before the SQL statement which uses the SHIFTLEFT function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

Shifts to the left by a given number of places (<<).

Parameter description:

- **`number1`**: Tinyint|Smallint|Int|Bigint integer.

- **number2**: An Int integer.

Return value:

Returns the Int or Bingint type.

Example:

```
shiftleft(1,2)=4  --Shifts the binary value of 1 two places to the
left (1<<2,0001 shifted to 0100)
shiftleft(4,3)=32  --Shifts the binary value of 4 three places to the
left (4<<3,0100 shifted to 10,0000)
```

SHIFTRIGHT

Function definition:

```
Int shiftright(Tinyint|Smallint|Int number1, Int number2)
Bigint shiftright(Bigint number1, Int number2)
```

Note:

Before the SQL statement which uses the SHIFTRIGHT function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

Usage:

This function is used for shifts right by a given number of places (>>).

Parameter description:

- **number1**: Tinyint|Smallint|Int|Bigint integer.
- **number2**: An Int integer.

Return value:

Returns the Int or Bigint type.

Example:

```
shiftright(4,2)=1 -- Shifts the binary value of 4 two places to the
right (4>>2,0100 shifted to 0001)
shiftright(32,3)=4 -- Shifts the binary value of 32 three places to
the right (32>>3,100000 shifted to 0100)
```

SHIFTRIGHTUNSIGNED

The command format is as follows:

```
Int shiftrightunsigned(Tinyint|Smallint|Int number1, Int number2)
```

```
Bigint shiftrightunsigned(Bigint number1, Int number2)
```

Note:

Before the SQL statement which uses the SHIFTRIGHTUNSIGNED function, add `set odps.sql.type.system.odps2=true;` to use the new data type function normally.

The command description is as follows:

This function is used for unsigned right shift by a given number of places (>>>).

Parameter description:

- **number1**: Tinyint|Smallint|Int|Bigint integer.
- **number2**: An Int integer.

Return value:

Returns the Int or Bigint type.

Example:

```
shiftrightunsigned(8,2)=2 -- Shifts the unsigned binary value of 8 two
 places to the right (8>>>2,1000 shifted to 0010)
shiftrightunsigned(-14,2)=1073741820  -- Shifts the unsigned binary
 value of -14 two places to the right (-14>>>2, 11111111 11111111
11111111 11110010 shifted to 00111111 11111111 11111111 11111100)
```

4.8.3 Window functions

In MaxCompute SQL, window functions help in analyzing and processing the workflow flexibly. Window function can only appear in the ‘select’ clause. However using both the nested window function and aggregate function in window function is not allowed. Also, it cannot be used at the same level as that of the aggregation function together.

Currently, in a MaxCompute SQL statement, you can use five window functions.

Window function syntax:

```
window_func() over (partition by [col1,col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] windowing_clause)
```

- **partition by** specifies open window columns. The rows of which partitioned columns have the same values are considered in the same window. Currently, a window can contain at most 100,000,000 rows data. We recommend that the rows must not exceed 5,000,000, otherwise, an error is reported at runtime.
- The clause **order by** specifies how the data is ordered in a window.

- In **windowing_clause** part, use rows to specify window open way. The two methods are as follows:
  - Rows between x preceding|following and y preceding|following, which indicates the window range is from rows x preceding /following to rows y preceding/following.
  - Rows x preceding|following: the window range is from rows x preceding /following to the present row.
  - 'x', 'y' must be an integer constant that is greater than or equal to 0 and corresponding value range is 0~10000. If the value is 0, it indicates the present row. Use the rows method to specify window range on condition that you have specified 'order by' clause for.

Note:

Not all window functions can be specified window open way using rows. The window functions support this usage include AVG, count, Max, min, StdDev, sum.

COUNT

Function definition:

```
Bigint count([distinct] expr) over(partition by [col1, col2…]
 [order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

Usage:

Calculates the total number of retrieved rows.

Parameter description:

- **expr**: Any data type. When it is NULL, this row is not counted. If the 'distinct' keyword is specified, it indicates using the unique count value.
- **partition by [col1, col2…]**: Specifies the columns to use window function.
- **order by col1 [asc|desc], col2[asc|desc]**: If 'order by' clause is not specified, return the count vale of 'expr' in the current window. If 'order by' clause is specified, the return result is ordered according to the specified sequence and the value is a cumulative count value from start row to the current row in the current window.

Return value:

Bigint type.

Note:

If the keyword 'distinct' is specified, the 'order by' clause cannot be used.

Example:

Thethe table ‘test_src’ already exists and the column ‘user_id’ of bigint type exists in this table.

```
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 |

    -- the ‘order by’ clause is not specified, return the count value
of user_id in the current window.
    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 | -- start row of the window
    | 1 | 2 | --two records exist from start row to current row.
Return 2.
    | 1 | 3 |
    | 2 | 1 |
    | 3 | 1 |

    -- The ‘order by’ clause is specified and return a cumulative
count value from start row to current row in the current window.
```

AVG

Function definition:

```
avg([distinct] expr) over(partition by [col1, col2…]
 [order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

Usage:

Calculates the average.

Parameter description:

- **`distinct`**: if the keyword ‘distinct’ is specified, it indicates taking average of the unique value.
- **`expr`**: Double type.
  - If the input is ‘string’ type or ‘bigint’ type, it is converted to ‘double’ type by implicit conversion and involved in the operation. If it is another data type, an exception is thrown.
  - If this value is NULL, then this row is not counted in the calculation.
  - If the data type is Boolean, then this row is excluded from the calculation.

- **`partition by [col1, col2...]`**: Specified the olumns to use window function.
- **`order by col1[asc|desc], col2[asc|desc]`**: If 'order by' clause is not specified, return the average of all values in the current window. If 'order by' clause is specified, the return result is ordered according to the specified sequence and returns the cumulative average from start row to current row in the current window.

Return value:

Double type.

Note:

If the keyword 'distinct' isn specified, the 'order by' clause cannot be used.

MAX

Function definition:

```
max([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

Usage:

Calculates the maximum value.

Parameter description:

- **`expr`**: Any types expect 'Boolean'. If the value is NULL, this row is not involved in the calculation. If the keyword 'distinct' is specified, it indicates taking the max value of the unique value.
- **`partition by [col1, col2…]`**: Specifies columns to use window function.
- **`order by [col1[asc|desc], col2[asc|desc`**: If 'order by' clause is not specified, return the maximum value in the current window. If 'order by' clause is specified, the return result is ordered according to the specified sequence and return the maximum value from start row to current row in the current window.

Return value:

Same as the 'expr' type..

Note:

If the keyword 'distinct' is specified, the 'order by' clause cannot be used.

MIN

Function definition:

```
min([distinct] expr) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause])
```

Usage:

Calculates the minimum value of the column.

Parameter description:

- **expr**Any types except 'Boolean'. If the value is NULL, this row is not counted in the calculation. If the keyword 'distinct' is specified, it indicates using the minimum value of a unique value.
- **partition by [col1, col2…]**: Specifies columns to use window function.
- **order by [col1[asc|desc], col2[asc|desc**: If 'order by' clause is not specified, return the minimum value in the current window. If 'order by' clause is specified, the return result is ordered according to the specified sequence and return the minimum value from start row to current row in the current window.

Return value:

the same type with 'expr'.

Note:

If the keyword 'distinct' is specified, the 'order by' clause cannot be used.

MEDIAN

Function definition:

```
Double median(Double number1,number2...) over(partition by [col1, col2
…])
Decimal median(Decimal number1,number2...) over(partition by [col1,
col2…])
```

Usage:

Calculates the median.

Parameter description:

- **number1,number1…**: 1 to 255 digits of a Double or Decimal type.
  - When the input value is a String type or a Bigint type, the operation is performed after the implicit conversion to a Double type, and other types throw exceptions.

- Return NULL when the input value is null.
- When the input value is a Double type, it converts to the Array of Double by default .

- **`partition by [col1, col2…]`**: Specifies columns to use window function.

Return value:

Double type.

STDDEV

Function definition:

```
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])
```

Usage:

Calculates population standard deviation.

Parameter description:

- **`expr`**: Double type.
  - If the input is ‘string’ or ‘bigint’ type, it is converted to ‘double’ type and is counted in the operation. If it is another data type, an exception is thrown.
  - If the input value is ‘NULL’, this row is excluded.
  - If the keyword ‘distinct’ is specified, it indicates calculating the population standard deviation of the unique value.
- **`partition by [col1, col2..]`**: Specifies columns to use window function.
- **`order by col1[asc|desc], col2[asc|desc]`**: If ‘order by’ clause is not specified, return the population standard deviation in the current window. If ‘order by’ clause is specified, the return result is ordered according to the specified sequence and return the population standard deviation from start row to current row in the current window.

Return value:

When the input is ‘decimal’ type, return ‘decimal’; otherwise, return ‘double’.

Example:

```
select window, seq, stddev_pop('1\01') over (partition by window order
 by seq) from dual;
```

Note:

- If the keyword ‘distinct’ is specified, the ‘order by’ clause cannot be used.
- **Stddev_pop** is an alias function of **stddev** function and its usage is the same as that of **stddev**

STDDEV_SAMP

Function definition:

```
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])
```

Usage:

Calculate sample standard deviation.

Parameter description:

- **Expr**: Double type.
  - If the input is ‘string’ or ‘bigint’ type, it is converted to ‘double’ type and counted in the operation. If it is another data type, an exception is indicated.
  - If the input value is NULL, this row is excluded.
  - If the keyword ‘distinct’ is specified, it indicates calculating the sample standard deviation of the unique value.
- **partition by [col1, col2..]**: Specifies columns to use window function.
- **Order by col1[asc|desc], col2[asc|desc]**: If ‘order by’ clause is not specified, return the sample standard deviation in the current window. If ‘order by’ clause is specified, the return result is ordered according to the specified sequence and return the sample standard deviation from start row to current row in the current window.

Return value:

When the input is ‘decimal’ type, return ‘decimal’; otherwise, return ‘double’.

Note:

If the keyword ‘distinct’ is specified, the ‘order by’ clause cannot be used.

SUM

Function definition:

```
sum([distinct] expr) over(partition by [col1, col2…]
```

```
[order by [col1[asc|desc], col2[asc|desc…]] [windowing_clause])
```

Usage:

Calculates the sum of elements.

Parameter description:

- **Expr**: Double type.
  - If the input is 'string' or 'bigint' type, it is converted to 'double' type and counted in the operation. If it is another data type, an exception is indicated.
  - If the input value is NULL, this row is excluded.
  - If the keyword 'distinct' is specified, it indicates calculating the sum of the unique value.
- **Partition by [col1, col2..]**: Specifies columns to use window function.
- **Order by col1[asc|desc], col2[asc|desc]**: If 'order by' clause is not specified, return the sum in the current window. If 'order by' clause is specified, the return result is ordered according to the specified sequence and return the sum from start row to current row in the current window.

Return value:

- If the input parameter is 'bigint' type, return 'bigint' type.
- If the input parameter is 'Decimal' type, return 'Decimal' type.
- If the input parameter is 'double' type or 'string' type, return 'double' type.

Note:
If the keyword 'distinct' is specified, the 'order by' clause cannot be used.

DENSE_RANK

Function definition:

```
Bigint dense_rank() over(partition by [col1, col2…]
order by [col1[asc|desc], col2[asc|desc]…])
```

Usage:

Calculates the dense rank. The data in the same row of col2 has the same rank.

Parameter description:

- **partition by [col1, col2..]**: Specifies columns to use window function.
- **order by col1[asc|desc], col2[asc|desc]**: Specifies the value which the rank is based on.

Return value:

Bigint type.

Example:

The data in table ‘emp’ is as follows:

```
| 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
```

Now, all employees need to be grouped by department, and each group must be sorted in descending order according to SAL to obtain the serial number in own group.

```
SELECT deptno
        , ename
        , sal
        , DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal DESC
) AS nums--Deptno as a window column, and sort in descending order
according to sal.
    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

Function definition:

```
Bigint rank() over(partition by [col1, col2…]
order by [col1[asc|desc], col2[asc|desc]…])
```

Usage:

Calculates the rank. The ranking of the same row data with col2 drops.

Parameter description:

- **`Partition by [col1, col2..]`**: Specifies columns to use window function.
- **`Order by col1[asc|desc], col2[asc|desc]`**: Specifies the value which the rank is based on.

Return value:

Bigint type.

Example:

The data in table ‘emp’ is as follows:

```
| 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
```

Now, all employees need to be grouped by department, and each group must be sorted in descending order according to SAL to obtain the serial number in own group.

```
SELECT deptno
        , ename
        , sal
        , RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS nums
--Deptno as a window column, and sort in descending order according to
 sal.
```

```
    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

Function definition:

```
lag(expr , Bigint offset, default) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]])
```

Command description:

Take the value of nth row in front of current row in accordance with offset. If the current row number is rn, take the value of the row which row number is rn-offset.

Parameter description:

- **`expr`**: Any type.
- **`offset`**: A Bigint type constant. If the input is String type or Double type, convert it to Bigint type by implicit conversion. Offset > 0;
- **`default`**: Define the default value while the specified range of ‘offset’ crosses the limit. It is constant and default is null.
- **`partition by [col1, col2..]`**: Specifies columns to use window function.
- **`order by col1[asc|desc], col2[asc|desc]`**: Specifies the order method for return result.

Return Value:

Returns the same with ‘expr’.

LEAD

Command format:

```
lead(expr , Bigint offset, default) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]])
```

Command description:

Take the value of nth row following current row in accordance with offset. If the current row number is rn, take the value of the row which row number is rn+offset.

Parameter description:

- **expr**: Any type.
- **offset**: A Bigint type constant. If the input is String, Decimal or Double type, convert it to Bigint type by implicit conversion. Offset > 0.
- **default**: Define the default value while the specified range of offset crosses the limit. It is constant.
- **partition by [col1, col2..]**: Specifies columns to use window function.
- **order by col1[asc|desc], col2[asc|desc]**: Specifies the order method for return result.

Return Value:

Same as the ‘expr’ type.

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

Command format:

```
Percent_rank () over (partition by [col1, col2...]
order by [col1[asc|desc], col2[asc|desc]…])
```

Command description:

Calculate relative ranking of a certain row in a group of data.

Parameter description:

- **partition by [col1, col2..]**: Specifies columns to use window function.
- **order by col1[asc|desc], col2[asc|desc]**: Specifies the value based on the ranking.

Return Value:

Returns the Double type, value scope is [0, 1]. The calculation method of relative ranking is `(rank -1)/(number of rows -1)`.

Note:

The current limit of rows in a single window cannot exceed 10,000,000.

ROW_NUMBER

Command format:

```
row_number() over(partition by [col1, col2…]
order by [col1[asc|desc], col2[asc|desc]…])
```

Command description:

Calculates the row number, beginning from 1.

Parameter description:

- **partition by [col1, col2..]**: Specifies columns to use window function.
- **order by col1[asc|desc], col2[asc|desc]**: Specifies the order method for return result.

Return Value:

Returns the Bigint type.

Example:

The data in table emp is as follows:

```
| 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, fig-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, fig-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, maid-12-30 00:00:00, 1300, 10
```

Now, all employees need to be grouped by department, and each group must be sorted in descending order according to SAL to obtain the serial number in own group.

```
SELECT deptno
        , ename
        , Sal
        , Row_number () over (partition by deptno order by Sal DESC
) as Nums  --Deptno as a window column, and sort in descending order
according to sal.
    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

Command format:

```
boolean cluster_sample([Bigint x, Bigint y])
over(partition by [col1, col2..])
```

Command description:

Used for Group sampling.

Parameter description:

- **x**: A Bigint type constant, x>=1. If you specify the parameter y, x indicates dividing a window into x parts. Otherwise x indicates selecting x rows records in a window (if x rows are in this window, return true). If x is NULL, return NULL.
- **y**: A Bigint type constant, y>=1, y<=x. It indicates selecting y parts records from x parts in a window (in other words, if y parts records exist, return value is true). If y is NULL, return NULL.

- **partition by [col1, col2]**: Specifies columns to use window function.

Return Value:

Returns the Boolean type.

Example:

If two columns key and value are in the table test_tbl, key is grouping field. The corresponding values of key have groupa and groupb, the field value indicates value of key shown as follows:

```
    | 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 |
```

To select 10% values from each group, the following MaxCompute SQL is recommended:

```
Select key, Value
    from (
        Select key, value, cluster_sample (10, 1) over (partition by
key) as flag
        from tbl
        ) sub
    where flag = true;

| Key | value |

| groupa | 0.167537127858695 |
| groupb | 0.135751224393789 |
```

NTILE

Command format:

```
BIGINT ntile(BIGINT n) over(partition by [col1, col2…]
```

```
[order by [col1[asc|desc], col2[asc|desc]…]] [windowing_clause]))
```

Command description:

Used to cut grouped data into N slices in order and return the current slice value, if the slice is uneven, the distribution of the first slice is increased by default.

Parameter description:

N: bigint data type.

Return Value:

Returns the bigint type.

Example:

The data in the table EMP is as follows:

```
| Empno | ename | job | Mgr | hiredate | Sal | REM | deptno |
7369, Smith, clerk, maid-12-17 00:00:00, 800, 20
7499, Allen, salesman, maid-02-20 00:00:00, 1600,300, 30
7521, Ward, salesman, maid-02-22 00:00:00, 1250,500, 30
7566, Jones, Manager, fig-04-02 00:00:00, 2975, 20
7654 Martin, salesman, fig-09-28 00:00:00, fig, 30
7698, Blake, Manager, fig-05-01 00:00:00, 2850, 30
7782, Clark, Manager, fig-06-09 00:00:00, 2450, 10
7788, Scott, analyst, fig-04-19 00:00:00, 3000, 20
00:00:00, King, President, 1991-11-17 5000, 7839, 10
7844, Turner, salesman, fig-09-08 00:00:00, 1500,0, 30
7876, Adams, clerk, maid-05-23 00:00:00, 1100, 20
7900 James, clerk, maid-12-03 00:00:00, 950, 30
7902 Ford, analyst, fig-12-03 00:00:00, 3000, 20
7934 Miller, clerk, fig-01-23 00:00:00, 1300, 10
7948, jaccka, clerk, fig-04-12 00:00:00, 5000, 10
7956, welan, clerk, fig-07-20 00:00:00, 2450, 10
7956, tebage, clerk, maid-12-30 00:00:00, 1300, 10
```

All employees now need to be divided into three groups according to Sal high to low cut, and get the serial number of the employee's own group.

```
Select deptno, ename, Sal, ntile (3) over (partition by depno order by
 Sal DESC) as nt3 from EMP;
-- Execution results as follows

| Deptno | ename | Sal | nt3 |

| 10 | jaccka | 5000.0 | 1 |
| 10 | King | 5000.0 | 1 |
| 10 | welan | 2450.0 | 2 |
| 10 | Clark | 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 | 2 |
| 20 | Smith | 800.0 | 3 |
```

```
| 30 | Blake | 2850.0 | 1 |
| 30 | Allen | 1600.0 | 1 |
| 30 | Turner | 1500.0 | 2 |
| 30 | Martin | 1250.0 | 2 |
| 30 | ward | 1250.0 | 3 |
| 30 | James | 950.0 | 3 |
```

4.8.4 Aggregate functions

The relation between the input and the output of aggregate functions is a many-to-one relationship; that is, to aggregate multiple input records into an output record. Use it with the group by clause in SQL.

COUNT

Function definition:

```
bigint count([distict|all] value)
```

Usage:

Counts the record numbers.

Parameter description:

- distinct|all: Specifies whether to remove duplicate records while counting. The default all counts all records. If the field ‘distinct’ is specified, then a unique count value is used.
- value: Any type. If the value is NULL, the corresponding row is not counted. Count (*), returns all rows.

Return Value:

Returns the Bigint type.

Example:

```
-- If the table tbla has the column col1 and the data type is Bigint.
+------+
| COL1 |
+------+
| 1 |
+------+
| 2 |
+------+
| NULL |
+------+
select count(*) from tbla;  -- value is 3.
select count(col1) from tbla;  -- value is 2
```

Use the aggregation function with the group by clause. Example, suppose that the table test_src has two columns, key is a String type, and value is a Double type.

```
-- The data of test_src is shown as follows:
```

```
+-----+-------+
| key | value |
+-----+-------+
| a | 2.0 |
+-----+-------+
| a | 4.0 |
+-----+-------+
| b | 1.0 |
+-----+-------+
| b | 3.0 |
+-----+-------+
-- Now run following sentence and get the result:
select key, count(value) as count from test_src group by key;
+-----+-------+
| key | count |
+-----+-------+
| a | 2 |
+-----+-------+
| b | 2 |
+-----+-------+
-- The aggregation function calculates the aggregate value that
has the same key value.The preceding rules apply to the following
aggregate functions also.
```

AVG

Function definition:

```
double avg(double value)
decimal avg(decimal value)
```

Usage:

Calculates the average value.

Parameter description:

value: Double or Decimal type. If the input is String or Bigint type, it is converted to Double type by implicit conversion. If it is another data type, an exception is thrown. If this value is NULL, a corresponding row is not counted in the calculation. The input cannot be Boolean type.

Return value:

If the input is Decimal type, then return Decimal type. If it is the other valid types, then return Double type.

Example:

```
-- If the table tbla has a column value and its data type is Bigint.
+-------+
| value |
+-------+
| 1 |
| 2 |
| NULL |
+-------+
-- the avg of this column is: (1+2)/2=1.5
```

```
select avg(value) as avg from tbla;
+------+
| avg |
+------+
| 1.5 |
+------+
```

MAX

Function definition:

```
max(value)
```

Usage:

Calculates the maximum value.

Parameter description:

value: Any data type. If the column value is NULL, the corresponding row is not counted in the operation. Values of the Boolean type are excluded from calculation.

Return value:

The return value is matchs the value type.

Example:

```
-- If the table tbla has a column clo1 and its data type is Bigint.
+------+
| col1 |
+------+
| 1 |
+------+
| 2 |
+------+
| NULL |
+------+
Select max (value) from tbla; -- return value is 2
```

MIN

Function definition:

```
MIN(value)
```

Usage:

Calculates the minimum value of the column.

Parameter description:

Any data type. If the column value is NULL, the corresponding row is not counted in the operation . A Boolean type is excluded from the operation.

Example:

```
-- If the table tbla has a column value and its data type is Bigint.
+------+
| value|
+------+
| 1 |
+------+
| 2 |
+------+

+------+
Select min (value) from tbla; -- return value is 1
```

MEDIAN

Function definition:

```
double median(double number)
decimal median(decimal number)
```

Usage:

Calculates the median.

Parameter description:

number: Double or Decimal type. If the input is String or Bigint type, it is converted to Double type and is counted in the operation. If it is another data type, an exception is thrown.

Return value:

Returns the Double or Decimal type.

Example:

```
-- If the table tbla has a column value and its data type is Bigint.
+------+
| value|
+------+
| 1 |
+------+
| 2 |
+------+
| 3 |
+------+
| 4 |
+------+
| 5 |
+------+
```

```
select MEDIAN(value) from tbla;  -- return value is 3.0
```

STDDEV

Function definition:

```
double stddev(double number)
decimal stddev(decimal number)
```

Usage:

Calculates a population standard deviation.

Parameter description:

number: Double type or Decimal type. If the input is String or Bigint type, it is converted to Double type and is counted in operation. If it is another data type, an exception is thrown.

Return value:

Returns a Double or Decimal type.

Example:

```
-- If the table tbla has a column value and its data type is Bigint.
+------+
| value|
+------+
| 1 |
+------+
| 2 |
+------+
| 3 |
+------+
| 4 |
+------+
| 5 |
+------+
select STDDEV(value) from tbla;  -- return value is 1.4142135623730951
```

STDDEV_SAMP

Function definition:

```
double stddev_samp(double number)
decimal stddev_samp(decimal number)
```

Usage:

Calculates a sample standard deviation.

Parameter description:

number: Double type or Decimal type. If the input is String or Bigint type, it is converted to Double type and is counted in operation. If it is another data type, an exception is thrown.

Return value:

Returns a Double or Decimal type.

Example:

```
-- If the table tbla has a column value and its data type is Bigint.
+------+
| value|
+------+
| 1 |
+------+
| 2 |
+------+
| 3 |
+------+
| 4 |
+------+
| 5 |
+------+
select STDDEV_SAMP(value) from tbla; -- return value is 1.5811388300
841898
```

SUM

Function definition:

```
sum(value)
```

Usage:

Calculates the sum of elements.

Parameter description:

value: Double, Decimal, or Bigint type. If the input is String type, it is converted to Double type and counted in operation. If the value in the column is NULL, this row is counted A Boolean type excluded from this calculation.

Return value:

If the input parameter is Bigint type, return Bigint type. If the input parameter is Double type or String type, return Double type.

Example:

```
-- If the table tbla has a column value and its data type is Bigint.
+------+
| value|
+------+
| 1 |
```

```
+------+
| 2 |
+------+
| NULL |
+------+
select sum(value) from tbla;   -- return value is 3
```

WM_CONCAT

Function definition:

```
string wm_concat(string separator, string str)
```

Usage:

Uses a specific separator to link the value in str.

Parameter description:

- · Separator: a String type constant. Constants of other types or non-constants can throw exceptions.
- Str: String type. If the input is String type, it is converted to Double type and is counted in operation. If it is another data type, an exception is thrown.

Return value:

Returns the String type.

Note:

For the sentence `select wm_concat(',', name) from test_src;`, if test_src is empty set, this MaxCompute SQL sentence returns NULL.

COLLECT_LIST

Function definition:

```
ARRAY collect_list(col)
```

Usage:

Within a given group, the expression specified by col is used to aggregate the data into an array.

Parameter description:

col: A table column can be any data type.

Return value:

Returns the ARRAY type.

Note:

Please add `set odps.sql.type.system.odps2=true;` in front of the SQL statement that uses this function, and submit it with SQL to use the new data type normally.

COLLECT_SET

Function definition:

```
ARRAY collect_set(col)
```

Usage:

Within a given group, the expression specified by col is used to aggregate the data into an array of non-repeating elements.

Parameter description:

col: A table column can be any data type.

Return value:

Return ARRAY type.

Note:

Please add `set odps.sql.type.system.odps2=true;` in front of the SQL statement that uses this function and submit it with SQL to use the new data type function normally.

4.8.5 String functions

This article introduces the string functions such as CHAR_MATCHCOUNT, CHR, CONCAT, GET_JSON_OBJECT, INSTR, IS_ENCODING supported by MaxCompute.

CHAR_MATCHCOUNT

Command format:

```
bigint char_matchcount(string str1, string str2)
```

Usage:

Calculates the total number of times each character in str1 is duplicated in str2.

Parameter description:

- str1, str2: String type, must be effective UTF-8 strings. If invalid character is in matching process, return a negative value.

- Return value: Bigint type, Any NULL input, return NULL.

Example:

```
char_matchcount('abd','aabc') = 2
-- Two strings 'a', 'b' in str1 appear in str2.
```

CHR

Command format:

```
string chr(bigint ascii)
```

Usage:

Convert the specified ASCII code ‘ascii’ into character.

Parameter description:

- ascii: Bigint type ASCII value. If the input is ‘string’ or ‘double’, it is converted to ‘bigint’ by implicit conversion. If the input is other types, an exception is thrown.
- Return value: String type. The parameter value range is [0,255]. An exception is thrown if exceeding this range. If the input is NULL, return NULL.

CONCAT

Command format:

```
string concat(string a, string b...)
```

Usage:

The return value is a result of connecting all strings.

Parameter description:

- a, b… String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String by implicit conversion. If the input is other types, an exception is thrown.
- String: Return value: String type. If no parameter exists or a certain parameter is NULL, return NULL.

Example:

```
concat('ab','c') = 'abc'
concat() = NULL
```

```
concat('a', null, 'b') = NULL
```

GET_JSON_OBJECT

Command format:

```
STRING GET_JSON_OBJECT(STRING json,STRING path)
```

Usage:

In a standard json string, the specified string is extracted according to the path.

Parameter description:

- json: String type, standard json format string.
- path: **String type, describing the path in json, starting with a dollor sign ($)**. For a description of the new implementation, see *JsonPath*.
  - $: Root object
  - . : Child operator
  - [] : Subscript operator for array
  - * : Wildcard for []
- String: Returns string type.

Note:

- Return NULL if json is null or invalid json format.
- Return NULL if path is null or invalid (does not exist in json).
- If json is valid and path also exists, the corresponding string is returned.

Example:

```
+----+
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"
}
```

Use the following query process to extract information in the JSON object:

```
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:

```
get_json_object('{"array":[["aaaa",1111],["bbbb",2222],["cccc",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

Command format:

```
bigint instr(string str1, string str2[, bigint start_position[, bigint
 nth_appearance]])
```

Usage:

Calculates where substring str2 is located in str1.

Parameter description:

- str1: String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String by implicit conversion. If the input is other types, an exception is thrown.
- str2: String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String by implicit conversion. If the input is other types, an exception is thrown.
- start_position: Bigint type, for other types, an exception is thrown. It indicates from which character of str1 a search must be started from and the default starting position is the first character position 1. If it is less than 0, it causes abnormality.
- nth_appearance：bigint type, greater than 0, represents position of the second match of a substring in the string. If the chain is of a different type or less than or equal to 0, an exception is thrown.
- Return value: Bigint type.

Note:

- If str2 is not found in str1, return 0.-
- If any input parameter is null, return null
- If str2 is NULL and always can be matched successfully, instr (‘abc’, ‘’) returns 1.

Example:

```
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

Command format:

```
boolean is_encoding(string str, string from_encoding, string
to_encoding)
```

Usage:

Determine whether the input string ‘str’ can be changed into a character set ‘to_encoding’ from a specified character set ‘from_encoding’. It can be used to Determine whether the input is garbled. The common use is to set ‘from_encoding’ to be ‘utf-8’ and ‘to_encoding’ to be ‘gbk’.

Parameter description:

- str: String type, if the input is NULL, return NULL. The empty string can be assumed to be belonged to any character set.
- from_encoding, to_encoding: String type, source, destination character sets. If the input is NULL, return NULL.
- Return value: Boolean type. If ‘str’ can be converted successfully, return true, otherwise, return false.

Example:

```
is_encoding('test', 'utf-8', 'gbk') = true
is_encoding('test', 'utf-8', 'gbk') = true
-- These two traditional Chinese characters are in GBK stock in China.
is_encoding('test', 'utf-8', 'gb2312') = false
-- The grapheme inventory of ‘GB2312’ does not contain these two
Chinese characters.
```

KEYVALUE

Command format:

```
KEYVALUE(STRING srcStr,STRING split1,STRING split2, STRING key)
KEYVALUE(STRING srcStr,STRING key) //split1 = ";" , split2 = ":"
```

Usage:

split ‘srcStr’ into ‘key-value’ pairs by split1 and separate ‘key-value’ pairs by split2. Return the value corresponding to key.

Parameter description:

- srcStr: Source string to be split.
- key: Specified to return the nth string. After the source string is split by ‘split1’ and ‘split2’, return the corresponding value according to the specification of the ‘key’ value.
- split1, split2: Strings used as delimiters by which ‘srcStr’ is split. If these two parameters are not specified in the expression, the default value of ‘split1’ is ’;’ and that of ‘split2’ is ’:’. If a string that has been split by split1 and has multiple split2, the return result is not defined.

Return value:

- String type.
- If ‘split1’ or ‘split2’ is NULL, return NULL.
- If ‘scrStr’ and ‘key’ are NULL or in case of no matched ‘key’, return NULL.
- If multiple ‘key-value’ matches, return the value corresponding to the first matched key.

Example 1:

```
keyvalue('0:1\;1:2', 1) = '2'
```

Note:

The source string is "0:1\;1:2". As split1 and split2 are not specified, the default split1 is ";" and split2 is ":".

After the split1 split, the key-value pair is 0:1\,1:2.

After split2 split, it becomes:

```
0 1/
1 2
```

Returns the value(2) of the key corresponding to 1.

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.
```

Note:

The source string is as follows:

```
“\;decreaseStore:1\;xcard:1\;isB2C:1\;tf:21910\;cart:1\;shipping:2\;
pf:0\;market:shoes\;instPayAmount:0\;”
```

The key-value pairs derived from the split after splitting according to the split1 '\;' are as follows:

```
decreaseStore:1 , xcard:1 , isB2C:1 , tf:21910 , cart:1 , shipping:2 , pf:0 ,
market:shoes , instPayAmount:0
```

After you split, follow the split2 ":", the results are as follows:

```
decreaseStore 1
 xcard 1
 isB2C 1
 tf 21910
 cart 1
 shipping 2
 pf 0
 market shoes
 instPayAmount 0
```

The value of the key parameter is "tf", the return value of the corresponding value parameter is 21910.

LENGTH

Command format:

```
bigint length(string str)
```

Usage:

Return the length of a string.

Parameter description:

- str: String type. If the input is Bigint，Double，Decimal or Datetime, it is converted to String by implicit conversion. If the input is other types, an exception is thrown.
- Return value: Bigint type. If ‘str’ is NULL, return NULL. If ‘str’ is non UTF-8 coding format, return -1.

Example:

```
length('hi! China') = 6
```

LENGTHB

Command format:

```
bigint lengthb(string str)
```

Usage:

Return the length of ‘str’ and its unit is byte.

Parameter description:

- str: String type. If the input is Bigint，Double，Decimal or Datetime, it is converted to String by implicit conversion. If the input is other types, an exception is thrown.
- Return value: Bigint type. If ‘str’ is NULL, return NULL.

Example:

```
lengthb('hi! 中国') = 10
```

MD5

Command format:

```
string md5(string value)
```

Usage:

Calculate the md5 value of input string.

Parameter description:

- value: String type. If the input value is of the Bigint, Double, Decimal or Datetime type, it is implicitly converted to the String type before calculation. If the input value is of another type, an exception is thrown. If the input is NULL, return NULL.
- Return value: String type.

REGEXP_EXTRACT

Command format:

```
string regexp_extract(string source, string pattern[, bigint
occurrence])
```

Usage:

Split the string source according to pattern (regular expression rules), and return the characters of the occurrence(nth) group.

Parameter description:

- source: String type, a string to be searched.
- pattern: A string type constant. If pattern is a null string, an exception is thrown. If ‘group’ is not specified in pattern, then also an exception is thrown.
- Occurrence: A bigint type constant, must be greater than 0 or equal to 0. If it is other type or less than 0, an exception is thrown. If not specified, the default value is 1, which indicates returning the first group. If ‘occurrence’ is equal to 0, then return substrings that satisfy the entire ‘pattern’.
- Return value: String type. Any input is NULL, return NULL.

Example:

```
regexp_extract('foothebar', 'foo(. *?)( bar)', 1) = the
regexp_extract('foothebar', 'foo(. *?)( bar)', 2) = bar
regexp_extract('foothebar', 'foo(. *?)( bar)', 0) = foothebar
regexp_extract('8d99d8', '8d(\\d+)d8') = 99
-- If regular SQL is submitted on MaxCompute, two "\" must be used as
the shift character.
regexp_extract('foothebar', 'foothebar')
-- The exception is thrown. ‘group’ is not specified in ‘pattern’.
```

REGEXP_INSTR

Function definition:

```
bigint regexp_instr(string source, string pattern[,
bigint start_position[, bigint nth_occurrence[, bigint return_option
]]])
```

Usage:

Returns the start position/end position of the substring, which matches the pattern with the source from start_position and nth_occurrence.. Any input parameter is null, return null.

Parameter description:

- source: String type, to be searched.
- pattern: A string type constant. If ‘pattern’ is null, an exception is thrown.
- start_position: Bigint type constant, the start position of search. If it is not specified, default value is 1. If it is other type or a value is less than or equal to 0, an exception is thrown.

- nth_occurrence: A bigint type constant. If not specified, the default value is 1. It appears at the first position, when searched. If it is less than or equal to 0 or other type, an exception is thrown .
- return_option: A bigint type constant. Its value is 0 or 1. If it is other type or an invalid value, an exception is thrown. 0 indicates returning the start position of the matched value. 1 indicates returning the end position of the matched value.
- Return value: Bigint type, the start or end position of a matched substring in source specified by return_option.

Example:

```
regexp_instr("i love www.taobao.com", "o[[:alpha:]]{1}", 3, 2) = 14
```

REGEXP_REPLACE

Command format:

```
string regexp_replace(string source, string pattern, string replace_st
ring[, bigint occurrence])
```

Usage:

replace the substring in source which is matched ‘pattern’ for nth occurrence to be a specified string ‘replace_string’ and then return.

Parameter description:

- source: String type, a string to be replaced.
- pattern: String type constant. The pattern to be matched. If it is null, an exception is thrown.
- replace_string: String type, the string after replacing matched pattern.
- occurrence: Bigint type constant, must be greater than or equal to 0. It indicates replacing nth matching to be replace_string. If it is 0, it indicates all matched substrings have been replaced. If it is other type or less than 0, an exception is thrown. It can be 0 by default.
- Return value: String type. When referencing a group which is not existent, do not replace the string. Returns NULL when the source, pattern, occurrence parameter is entered as null, returns NULL, replace_string is null, but pattern will not match, if the replace_string is null and the pattern is matched, returns the original string.

Note:

When the reference group does not exist, it is considered to be undefined.

Example:

```
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 a group is defined in pattern and the referenced second group
is not existent.
-- Please avoid this. The result to reference nonexistent group is not
 defined.
regexp_replace("abcd", "(. *)(.)$", "\\2", 0) = "d"
regexp_replace("abcd", "a", "\\1", 0) = "bcd"
-- No group definition is in pattern, so '\1' references a nonexistent
 group,
 -- Please avoid this. The result to reference nonexistent group is
not defined.
```

REGEXP_SUBSTR

Command format:

```
string regexp_substr(string source, string pattern[, bigint start_posi
tion[, bigint nth_occurrence]])
```

Usage:

Starting from start_position, find a substring in source which matches with a specified pattern for the nth occurrence.

Parameter description:

- source: String type, string to be searched.
- pattern: A string type constant. The pattern to be matched. If it is null, an exception is thrown.
- start_position: A Bigint type constant, must be greater than 0. Other types or less than equal to 0 throw exceptions. If not specified the default value is 1, which indicates a match begins with the first character of source. If not specified, default value is 1. It indicates a matching value from the first character of source.
- nth_occurrence: a Bigint type constant, must be greater than 0. If not specified, the default value is 1. It indicates the return substring of the first matched value. If not specified, the default value is 1. It indicates the return substring of the first matched value.
- Return value: String type. Any input parameter is NULL, return NULL. If no matching record exists, return NULL.

Example:

```
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

Command format:

```
bigint regexp_count(string source, string pattern[, bigint start_posi
tion])
```

Usage:

Counts the number of occurrences that a substring matches with a specified pattern, starting from start_position in source.

Parameter description:

- Source: String type, the string to be searched. If it is the other type, an exception is thrown.
- Pattern: String type constant, the pattern to be matched. If it is a null string or other data type, an exception is thrown.
- start_position: Bigint type constant, must be greater than 0. If it is other data type or a value which is less than or equal to 0, an exception is thrown. If not specified, default value is 1, which indicates a matched value from the first character of source.
- Return value: Bigint type. If matching does not exists, return 0. If any input parameter is null, return null.

Example:

```
regexp_count('abababc', 'a.c') = 1
regexp_count('abcde', '[[:alpha:]]{2}', 3) = 1
```

SPLIT_PART

Command format:

```
string split_part(string str, string separator, bigint start[, bigint
end])
```

Usage:

Split the string str according to the separator and return the substring from nth start part to nth end part.

Parameter description:

- str: String type, the string to be split. If it is Bigint, Double, Decimal or Datetime, it is converted to a String in an implicit conversion. If it is other data type, an exception is thrown.
- separator: A string type constant, the separator used to split the string. It can be a character or a string. If it is other data type, an exception is thrown.
- start: A bigint type constant, must be greater than 0. If it is not a constant or other data type, an exception is thrown. It indicates the start number of the return part (start from 1). If the end is not specified, returns the part specified by 'start'.
- 'end': A bigint type constant, must be greater than or equal to 'start', otherwise an exception is thrown. It refers to the end number of the return part. If it is not a constant or is other data type, then also an exception is thrown. It can be excluded as it indicates the last part.

Return value: String type. If any parameter is null, return null. If separator is an empty string, return the source string str.

Note:

- If 'delimiter' does not exist in str, then specify 'start' as 1, and return the entire str. If the input value is an empty string, the output value is an empty string.
- If the start value is greater than the number of parts after split, for example, the split produces 6 parts but the 'start' value is greater than 6, then returns an empty string.

Example:

```
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

Command format:

```
string substr(string str, bigint start_position[, bigint length])
```

Usage:

Returns a substring of 'str' from start_position with the given length.

Parameter description:

- 'str': String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String in an implicit conversion. If it is other data type, an exception is thrown.
- The start_position:Bigint type starts at 1. Returns empty strings when start_position is 0.When start_position is negative, the starting position is counted backwards from the end of the

string, the last character is -1, and the previous number is -2,-3 and so on. Other types throw exceptions.

- length: Bigint type, must be greater than 0. If it is other type or less than 0, an exception is thrown. This parameter indicates the length of a child string.
- Return value: String type. If the input is NULL, return NULL.

Note:

If the length is excluded, return the substring from start to end.

Example:

```
substr("abc", 2) = "bc"
substr("abc", 2, 1) = "b"
substr("abc",-2,2)="bc"
substr("abc",-3)="abc"
```

SUBSTRING

Command format:

```
string  substring(string|binary str, int start_position[, int length])
```

Usage:

Returns the substring of ‘str’ from start_position with the given length.

Parameter description:

- str: String or Binary type, returns NULL or throws an exception for the other type
- ‘start_position’: Int type, starting at 1. Returns empty strings when start_position is 0.When start_position is negative, the starting position is counted backwards from the end of the string , the last character is-1, and the previous number is in turn-2,-3 and so on. Other types throw exceptions.
- length: Bigint type, must be greater than 0. If it is other type or less than 0, an exception is thrown. This parameter indicates the length of the child string.
- Return value: String type. If the input is NULL, return NULL.

Note:

If the length is excluded, return the substring from start to end.

For example:

```
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

Command format:

```
string tolower(string source)
```

Usage:

Input the lowercase string corresponding to the English string source.

Parameter description:

- Source: String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String in an implicit conversion. If it is other data type, an exception is throwm.
- Return Value: String type. If the input is NULL, return NULL.

Example:

```
tolower("aBcd") = "abcd"
tolower("Haha Cd") = "haha cd"
```

TOUPPER

Command format:

```
string toupper(string source)
```

Usage:

Output the uppercase string corresponding to the English string ‘source’.

Parameter description:

- Source: String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String in an implicit conversion. If it is other data type, an exception is throwm.
- Return Value: String type. If the input is NULL, return NULL.

Example:

```
toupper("aBcd") = "ABCD"
toupper("HahaCd") = "HAHACD"
```

TO_CHAR

Command format:

```
string to_char(boolean value)
string to_char(bigint value)
```

```
string to_char(double value)
string to_char(decimal value)
```

Usage:

Convert Boolean type, Bigint type or Double type to corresponding String type.

Parameter description:

- Value: Boolean, Bigint or Double type is acceptable. If it is other data type, an exception is thrown. For formatted output of the datetime type, see another function TO_CHAR that has the same name.
- Return value: String type. If the input is NULL, return NULL.

Example:

```
to_char(123) = '123'
to_char(true) = 'TRUE'
to_char(1.23) = '1.23'
to_char(null) = NULL
```

TRIM

Command format:

```
string trim(string str)
```

Usage:

Removes left space and right space for the input string str.

Parameter description:

- ‘str’: String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String in an implicit conversion. If it is other data type, an exception is throwm.
- Return value: String type. If the input is NULL, return NULL.

LTRIM

Command format:

```
string ltrim(string str)
```

Usage:

Removes the left space for the input string str.

Parameter description:

- ‘str’: String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String in an implicit conversion. If it is other data type, an exception is throwm.
- Return value: String type. If the input is NULL, return NULL.

Example:

```
select ltrim(' abc ') from dual;
Returns:
+-----+
| _c0 |
+-----+
| abc |
+-----+
```

RTRIM

Command format:

```
string rtrim(string str)
```

Usage:

Removes the right space for the input string str.

Parameter description:

- ‘str’: String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String in an implicit conversion. If it is other data type, an exception is throwm.
- Return value: String type. If the input is NULL, return NULL.

Example:

```
select rtrim('a abc ') from dual;
Returns:
+-----+
| _c0 |
+-----+
| a abc |
+-----+
```

REVERSE

Command format:

```
STRING REVERSE(string str)
```

Usage:

Returns a reversed-order string.

Parameter description:

- str: String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String in an implicit conversion. If it is other data type, an exception is throwm.
- Return value: String type. If the input is NULL, return NULL.

Example:

```
select reverse('abcedfg') from dual;
Returns:
+-----+
| _c0 |
+-----+
| gfdecba |
+-----+
```

SPACE

Command format:

```
STRING SPACE(bigint n)
```

Usage:

A space string function that returns a string of length n.

Parameter description:

- n: Bigint type. The length cannot exceed 2 MB. If it is NULL, an exception is thrown.
- Return value: String type.

Example:

```
select length(space(10)) from dual; ----Returns 10.
select space(400000000000) from dual; ----Error, the length exceeds 2
MB.
```

REPEAT

Command format:

```
STRING REPEAT(string str, bigint n)
```

Usage:

Returns the str string that is repeated for n times.

Parameter description:

- ‘str’: String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String in an implicit conversion. If it is other data type, an exception is throwm.
- n: Bigint type. The length does not exceed 2 MB. If it is NULL, an exception is thrown.

- Return value: String type.

Example:

```
select repeat('abc',5) from lxw_dual;
Returns:abcabcabcabcabc
```

ASCII

Command format:

```
Bigint ASCII(string str)
```

Usage:

Returns the ascii of the first character of str.

Parameter description:

- str: String type. If the input is Bigint, Double, Decimal or Datetime, it is converted to String in an implicit conversion. If it is other data type, an exception is throwm.
- Return value: Bigint type.

Example:

```
select ascii('abcde') from dual;
Returns:97
```

Maxcomputerte2.0 Extension function

With the upgrade to MaxCompute 2.0, some mathematical functions have been added to the product. If a new function uses a new data type, you must add the following set statement before using the new functions SQL statement:

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

Note:

Add set odps before the SQL statement that uses the function`set odps.sql.type.system.odps2 = true;`, and commit runs with SQL to use the new data type function normally.

The enhanced and extended string functions are described as follows.

CONCAT_WS

Command format:

```
string concat_ws(string SEP, string a, string b...)
```

```
string concat_ws(string SEP, array)
```

Note:

Add set odps before the SQL statement that uses the function`set odps.sql.type.system.odps2 = true;`, and commit runs with SQL to use the new data type function normally.

Usage:

Concatenates all strings in the parameters, connected by the specified delimiter.

Parameter description:

- SEP: String-type delimiter. If not specified, an exception is returned.

Return value:

String type. If no parameters exist or any parameter is null, return null.

Example:

```
concat_ws(':','name','hanmeimei')='name:hanmeimei'
concat_ws(':','avg',null,'34')=null
```

LPAD

Command format:

```
string lpad(string a, int len, string b)
```

Note:

Add set odps before the SQL statement that uses the function`set odps.sql.type.system.odps2 = true;`, and commit runs with SQL to use the new data type function normally.

Usage:

Uses string b to pad string a to the left to the place specified by len.

Parameter description:

- len: Int-type integer.
- a/b…: String type.

Return value:

String type. If len is smaller than the number of places in a, a is truncated from the left to obtain a string with the number of places specified by len. If len is 0, return empty.

Example:

```
lpad('abcdefgh',10,'12')='12abcdefgh'
lpad('abcdefgh',5,'12')='abcde'
lpad('abcdefgh',0,'12') Returns a blank result
```

RPAD

Command format:

```
string rpad(string a, int len, string b)
```

Note:

Add set odps before the SQL statement that uses the function`set odps.sql.type.system.odps2 = true;`, and commit runs with SQL to use the new data type function normally.

Usage:

Uses string b to pad string a to the right to the place specified in len.

Note:

You need to add the set odps statement before the SQL statement that uses the function`set odps.sql.type.system.odps2 = true`, otherwise the error is reported.

Parameter description:

- len: Int-type integer.
- a/b…: String type.

Return value:

String type. If len is smaller than the number of places in a, a is truncated from the left to obtain a string with the number of places specified by len. If len is 0, return empty.

Example:

```
rpad('abcdefgh',10,'12')='abcdefgh12'
rpad('abcdefgh',5,'12')='abcde'
```

```
rpad('abcdefgh',0,'12') Returns a blank result
```

REPLACE

Command format:

```
string replace(string a, string OLD, string NEW)
```

Note:

Add set odps before the SQL statement that uses the function`set odps.sql.type.system.odps2 = true;`, and commit runs with SQL to use the new data type function normally.

Usage:

Uses string NEW to replace the portion of string a that completely matches string OLD and returns string a.

Parameter description:

The parameters are all String type.

Return value:

String type. If the input is null, return null.

Example:

```
replace('ababab','abab','12')='12ab'
replace('ababab','cdf','123')='ababab'
replace('123abab456ab',null,'abab')=null
```

SOUNDEX

Command format:

```
string soundex(string a)
```

Note:

Add set odps before the SQL statement that uses the function`set odps.sql.type.system.odps2 = true;`, and commit runs with SQL to use the new data type function normally.

Usage:

Converts a normal string to a soundex string.

Parameter description: a is of type String.

Return value: String type. If the input value is NULL, return NULL.

Example:

```
soundex('hello')='H400'
```

SUBSTRING_INDEX

Command format:

```
string substring_index(string a, string SEP, int count))
```

Note:

Add set odps before the SQL statement that uses the function`set odps.sql.type.system.odps2 = true;` and commit runs with SQL to use the new data type function normally.

Usage:

Truncates string a to the portion in front of the delimiter specified by count. If count is positive, the portion to the left of the delimiter is used. If count is negative, the portion to the right is used.

Parameter description: a/sep belong to the string type, and count belongs to the int type.

Return value:

String type. If the input is null, return null.

Example:-

```
substring_index('https://help.aliyun.com', '.', 2)='https://help.
aliyun'
substring_index('https://help.aliyun.com', '.', -2)='aliyun.com'
substring_index('https://help.aliyun.com', null, 2)=null
```

4.8.6 Other functions

This article shows you how to use functions such as cast, decode, least, array, split, map, and so on.

CAST

Function definition:

```
cast(expr as <type>)
```

Convert the result of expression to object type. For example, cast (‘1’ as bigint) is to convert string ‘1’ to bingint ‘1’. If the conversion is unsuccessful or the conversion is not supported, an exception occurs.

Note:

- cast (double as bigint) converts double type value to bigint type value.
- cast(string as bigint) converts a value of the string type into a value of the bigint type. If the string is composed of numerals expressed in integer form, it is directly converted into a value of the bigint type.
- If the string is composed of numerals expressed in the float or exponent form, it will be converted into a value of the double type first and then into a value of the bigint type.
- cast(string as datetime) or cast(datetime as string) adopts the default format `yyyy-mm-dd hh:mi:ss`.

COALESCE

Function definition:

```
coalesce(expr1, expr2, ...)
```

Usage:

Return the first value which is not NULL from the list. If all values in the list are NULL, return NULL.

Parameter description:

expr: value to be tested. All these values have the same data type or be NULL, otherwise an expection occurs.

Return value:

Return value type is the same as parameter type.

Note:
There must be one parameter at least, otherwise an exception occurs.

DECODE

Function definition:

```
decode(expression, search, result[, search, result]...[, default])
```

Usage:

Implement the selection function of if-then-else branch.

Parameter description:

- expression: expression to be compared.
- search: A search string to be compared with the expression.

- result: the returned value when the values of search and expression match.
- default: it is optional. If all search items do not match the expression, return this default value. If it is not specified, return NULL.

Return Value:

- return matched search.
- If no matching record exists, return default.
- If default is not specified, return NULL.

> **Note:**
>
> - You must specify at least three parameters.
> - All of the result types must be the same or NULL. Inconsistent data type causes an exception. All of the search and expression types must be consistent, otherwise an exception is reported.
> - If the option search in decode has repeated record and has been matched, return the first value.

Example:

```
Select
decode(customer_id,
1, 'Taobao',
2, 'Alipay',
3, 'Aliyun',
Null, 'N/A',
'Others') as result
from sale_detail;
```

The decode function mentioned previously implements the function in following if-then-else sentence:

```
if customer_id = 1 then
result := 'Taobao';
elsif customer_id = 2 then
result := 'Alipay';
elsif customer_id = 3 then
result := 'Aliyun';
...
else
result := 'Others';
end if;
```

> **Note:**

- Calculating NULL= NULL by MaxCompute SQL, return NULL, while the values of NULL and NULL are equal in decode function.
- In the preceding example, if the value of **customer_id** is NULL, decode function returns N/A as a result.

GET_IDCARD_AGE

Function definition:

```
get_idcard_age(idcardno)
```

Usage:

Returns the current age based on the ID number which is the difference of the current year and the year of birth identified in the ID.

Parameter description:

idcardno: String type, ID number of 15-digit or 18-digit. In the calculation, the validity of the ID is checked according to the province code and the last digit, and Null is returned if the check fails.

Return Value:

Returns the Bigint type. Input is Null, returns Null. Returns Null if the difference of the current year and the year of birth is larger than 100.

GET_IDCARD_BIRTHDAY

Function definition:

```
get_idcard_birthday(idcardno)
```

Usage:

Returns date of birth based on the ID number.

Parameter description:

idcardno: String type, ID number of 15-digit or 18-digit. In the calculation, the validity of the ID is checked according to the province code and the last digit, and Null is returned if the check fails.

Return Value:

Returns the Datetime type. Input is Null, returns Null.

GET_IDCARD_SEX

Function definition:

```
get_idcard_sex(idcardno)
```

Usage:

Returns the gender based on the ID number and the value is either M (male) or F (female).

Parameter description:

idcardno: String type, ID number of 15-digit or 18-digit. In the calculation, the validity of the ID is checked according to the province code and the last digit, and Null is returned if the check fails.

Return Value:

Returns the String type. Input is Null, returns Null.

GREATEST

Function definition:

```
greatest(var1, var2, ...)
```

Usage:

Return the greatest input parameter.

Parameter description:

var1/var2: Its type can be Bigint, Double, Decimal，Datetime or String type. If all values are NULL , return NULL.

Return Value:

- The greatest value in input parameter. If the implicit conversion is not needed, return type is the same as input parameter type.
- NULL is the least value.

  If the input parameter types are different,

  - For Double, Bigint, Decimal and String type, convert them to be Double type.
  - For String and Datetime, convert them to be Datetime type.
  - Other implicit conversion is not allowed.

ORDINAL

Function definition:

```
ordinal(bigint nth, var1, var2, ...)
```

Usage:

Return the location value specified by ‘nth’ after the input variables are sorted by small to large.

Parameter description:

- nth: Bigint type, specify the location to return its value. If it is NULL, return NULL.
- var1/var2: Its type can be Bigint, Double, Datetime or String type.

Return Value:

- The value in nth bit. If the implicit conversion is not needed, return type is the same as input parameter type.
- If implicit conversion is in input parameters,
  - For Double, Bigint and String type, convert them to be Double type.
  - For String and Datetime type, convert them to be Datetime type.
  - Other implicit conversion is not allowed.
- NULL is the least value.

Example:

```
ordinal(3, 1, 3, 2, 5, 2, 4, 6) = 2
```

LEAST

Function definition:

```
least(var1, var2, ...)
```

Usage:

return the least value in input parameter.

Parameter description:

var1/var2: Its type can be Bigint, Double, Decimal，Datetime or String type. If all values are NULL , return NULL.

Return Value:

- The least value in input parameter; If the implicit conversion is not needed, return type is the same as input parameter type.
- If implicit conversion is in input parameters,
  - For Double, Bigint and String type, convert them to be Double type.
  - For ‘string’ type and ‘datetime’ type, convert them to be ‘datetime’ type.
  - Converts to Decimal type when Decimal type compares to Double, Bigint or String type.
  - Other implicit conversion is not allowed.
- NULL is the least value.

MAX_PT

Function definition:

```
max_pt(table_full_name)
```

Usage:

For a partitioned table, this function returns the maximum value of the level-one partition of the partitioned table, which is sorted alphabetically, and there is a corresponding data file for the partition.

Parameter description:

table_full_name: String type, specifys the name of table, which must be with the name of project, for example: prj.src). You must own read permission on this table.

Return Value:

Return value: Returns the value of the largest level-one partition.

Example:

Example: Suppose that ‘tbl’ is a partitioned table, all partitions of the table are as follows, and there are data files:

```
pt =‘20120901’
pt =‘20120902’
```

In the following statement, the return value of **max_pt** is ‘20120902’, and the MaxCompute SQL statement reads the data in the ‘20120902’ partition.

```
select * from tbl where pt=max_pt('myproject.tbl');
```

Note:

If a new partition is added by using alter table, but there is no data file in this partition, then this partition is not returned.

UUID

Function definition:

```
string uuid()
```

Usage:

Return a random ID. Example: `29347a88-1e57-41ae-bb68-a9edbdd94212`.

Note:
UUID returns a random global ID with a low probability of duplication.

SAMPLE

Function definition:

```
boolean sample(x, y, column_name)
```

Usage:

sample all values of column_name according to the setting of x and y and filter out the rows which do not meet the sampling condition.

Parameter description:

- x, y: Bigint type, indicates hash to x portions, take yth portions. y can be ignored.
  - If y is ignored, take the first portion. If y in parameter is ignored, then column_name is ignored at the same time.
  - x and y are Bigint constants and greater than 0. If it is other data type or less than or equal to 0, an exception is thrown. If y&gt, x exception is also thrown. If any input of x and y is NULL, return NULL.
- column_name: the destination column to be sampled.
  - column_name can be omitted, in which case, a random sample is taken according to the values of x and y.
  - It can be any data type and the column value can be NULL. Do not need implicit type conversion.
  - If column_name is the constant NULL, an exception is reported.

Return Value:

Boolean type.

Note:

To avoid data skew brought by NULL value, NULL values in column_name will be carried out a uniform hash in x portions. If column_name is not added, the output is not necessarily uniform since the data size is smaller. So column_name is suggested to be added to get better output.

Example:

Suppose that the table tbla is existent and a column cola is in this table:

```
select * from tbla where sample (4, 1 , cola) = true;
-- The values are carried out Hash into 4 portions and take the first
portion.
select * from tbla where sample (4, 2) = true;
-- The values do random Hash into 4 portions for each row of data and
take the second portion.
```

CASE WHEN EXPRESSION

MaxCompute provides two kinds of case when syntax formats, as follows:

```
case
when (_condition1) then result1
when (_condition2) then result2
...
else resultn
end
case
when (_condition1) then result1
when (_condition2) then result2
when (_condition3) then result3
...
else resultn
end
```

Case when expression can return different values according to the computing result of expression values flexibly.

The following sentences is used to get the region according to different 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 result include Bigint and Double, convert them to Double type and then return the result.
- If the types of result include string type, convert them to be string type and then return the result. If the conversion is unsuccessfully, the error is reported. (such as Boolean type).
- Expect these, the conversion between other types is not allowed.

If expression

Function definition:

```
if(testCondition, valueTrue, valueFalseOrNull)
```

Usage:

Judge if testCondition is true. If it is true, return valueTrue, otherwise return valueFalse or Null.

Parameter description:

- testCondition: The expression to be judged. Boolean type.
- valueTrue: It returns when the expression testCondition is true.
- valueFalseOrNull: It returns when the expression testCondition is not true and also can be null.

Return Value:

The return type is the same as the valueTrue or valueFalseOrNul type.

Example:

```
select if(1=2,100,200) from dual;
--Returned results:
+ ------------ +
| _c0 |
+------------+
| 200 |
+------------+
```

SPLIT

Function definition:

```
split(str, pat)
```

Purpose: After the STR is split by Pat, the array is returned.

Parameter description:

- str: String type, specifies the string to be separated.
- pat: String type, specifies the delimiter, supports regular expressions.

Return Value:

`array <string>`

The result is the elements in str separated by pat.

Example:

```
select split("a,b,c",",") from dual;
Results:
+------+
| _c0 |
+------+
| [a, b, c] |
+------+
```

Note:

Set commands supported by MaxCompute SQL and MapReduce for MaxCompute 2.0

- Once data type such as Tinyint、Smallint、 Int、 Float、Varchar or TIMESTAMP BINARY is involved when running an SQL statement, `set odps.sql.type.system.odps2=true;` must be added before the SQL statement. The set statement and SQL statement are submitted simultaneously.
- Project level: that is, the project level is supported for new type opening. The project owner can be set to project as needed, with the following commands:

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

STR_TO_MAP

Function declaration:

```
str_to_map(text [, delimiter1 [, delimiter2]])
```

Purpose: use ‘delimiter1’ to separate ‘text’ into K-V pairs, then use ‘delimiter2’ to separate each K-V pair.

Parameter description

- text: String type, specifies the string to be separated.
- delimiter1: string type, separator that does not specify the default ','.
- delimiter1: string type, separator, default to '=' when not specified '.

Return value: map <string, string >. The elements are the K-V results of the separation of 'text' by the strings 'delimiter1' and 'delimiter2'.

Example:

```
Select fig ('test1 &amp; 1-test2 & 2 ','-','&amp;');
```

Return result:

```
+------------+
| A |
+------------+
| {Test1: 1, Test2: 2} |
```

EXPLODE

Function definition:

```
explode(var)
```

Usage:

Converts one row of data into a multi-row UDTF.

- If var is Array type, the array stored in the column is converted to multiple rows.
- If var is Map type, each key-value pair of the map stored in the column is converted to a row with two columns, one column for the key and one for the value.

Parameter description:

`var: array<T> type or map<K, V> type.`

Return Value:

Rows after conversion are returned.

Note:

The following restrictions apply when using UDTF:

- One select can only have one UDTF and no other columns can appear.
- It cannot be used with group by, cluster by, distribute by, or sort by.

Example:

```
explode(array(null, 'a', 'b', 'c')) col
```

MAP

Function definition:

```
MAP map(K key1, V value1, K key2, V value2, ...)
```

Usage:

Uses the given key/value pairs to create a map.

Parameter descriptio:

key/value

- All key types are consistent, including those after implicit conversion, and must be basic.
- All value types are consistent, including those after implicit conversion, and can be of any type.

Return Value:

Returns the map type.

Example:

For example, the fields in t_table are(c1 bigint,c2 string,c3 string, c4 bigint ,c5 bigint), with the following data

```
+------------+----+----+------------+------------+
| c1         | c2 | c3 | c4         | c5         |
+------------+----+----+------------+------------+
| 1000       | k11 | k21 | 86         | 15         |
| 1001       | k12 | k22 | 97         | 2          |
| 1002       | k13 | k23 | 99         | 1          |
+------------+----+----+------------+------------+
```

Execute SQL:

```
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} |
```

```
+------+
```

MAP_KEYS

Function definition:

```
ARRAY map_keys(map<K, V>)
```

Usage:

Returns an array of all the keys in the map parameter.

Parameter description:

map：map type data.

Return value:

Returns the array type, enter null, and null.

Example:

For example, the field of **t_table_map** is (`c1 bigint,t_map map<string,bigint>`), data as follows

```
+------------+-------+
| C1 | t_map |
+ ------------ + ------- +
| 1000       | {k11:86, k21:15} |
| 1001       | {k12:97, k22:2} |
| 1002       | {k13:99, k23:1} |
+------------+-------+
```

Execute SQL:

```
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 definition:

```
ARRAY map_values(map<K, V>)
```

Usage:

Returns an array of all the values in the map parameter.

Parameter description:

map: map-type data.

Return Value:

Returns the array type, enter null, and null.

Example:

```
select map_values(map('a',123,'b',456));
Results:
[123, 456]
```

ARRAY

Function definition:

```
ARRAY array(value1,value2, ...)
```

Usage:

Creates an array using the given values.

Parameter description:

value: This parameter can be of any type, but all the values must be of the same type.

Return Value:

Returns the array type.

Example:

For example, the fields in **t_table** are (c1 bigint,c2 string,c3 string, c4 bigint ,c5 bigint), with the following data

```
+------------+----+----+------------+------------+
| c1         | c2 | c3 | c4         | c5         |
+------------+----+----+------------+------------+
| 1000       | k11 | k21 | 86         | 15         |
| 1001       | k12 | k22 | 97         | 2          |
| 1002       | k13 | k23 | 99         | 1          |
```

```
+------------+----+----+-----------+-----------+
```

Execute SQL:

```
select array(c2,c4,c3,c5) from t_table;
```

Results:

```
+------+
| _c0 |
+------+
| [k11, 86, k21, 15] |
| [k12, 97, k22, 2] |
| [k13, 99, k23, 1] |
+ ---- +
```

SIZE

Function definition:

```
INT size(map)
INT size(array)
```

Usage:

- `size(map<K , V>)` returns the number of K/V pairs in the given map.
- `size(array<T>)` returns the number of elements in the given array.

Parameter description:

- `map<K, V>`: Map-type data.
- `array<T>`: Array-type data.

Return Value:

Returns the Int type.

Example:

```
select size(map('a',123,'b',456)) from dual;--Returns 2
select size(map('a',123,'b',456,'c',789)) from dual;--Returns 3
select size(array('a','b')) from dual;--Returns 2
select size(array(123,456,789)) from dual;--Returns 3
```

ARRAY_CONTAINS

Function definition:

```
boolean array_contains(ARRAY<T> a,value v)
```

Usage:

Checks if the given array a contains v.

Parameter description:

- a: Array-type data.
- v: The given v must be of the same type as the data in the array.

Return Value:

Returns the Boolean type.

Example:

If the field of **t_table_array** is (`c1 bigint, t_array array<string>`), the data is as follows:

```
+ ------------ + --------- +
| c1         | t_array |
+------------+---------+
| 1000       | [k11, 86, k21, 15] |
| 1001       | [k12, 97, k22, 2] |
| 1002       | [k13, 99, k23, 1] |
+------------+---------+
```

Execute SQL:

```
select c1, array_contains(t_array,'1') from t_table_array;
```

Results:

```
+------------+------+
| c1         | _c1  |
+------------+------+
| 1000       | false |
| 1001       | false |
| 1002       | true |
+------------+------+
```

SORT_ARRAY

Function definition:

```
ARRAY sort_array(ARRAY<T>)
```

Usage:

This function used to sorts the given array.

Parameter description:

`ARRAY<T>`: Array-type data, the data in the array can be of any type.

Return Value:

Returns the array type.

Example:

```
select sort_array(array('a','c','f','b')),sort_array(array(4,5,7,2,5,8
)),sort_array(array('You','Me','He')) from dual;
Results:
[a, b, c, f] [2, 4, 5, 5, 7, 8] [He, You, Me]
```

Execute SQL

```
Select sort_array (C1), sort_array (C2), sort_array (C3) from t_array;
```

Return result:

```
[a, b, c, f] [2, 4, 5, 5, 7, 8] [He, You, Me]
```

POSEXPLODE

Function definition:

```
posexplode(ARRAY<T>)
```

Usage:

Explodes the given array. Each value is given a row and each row has two columns corresponding to the subscript (starting from 0) and the array element.

Parameter description:

ARRAY: Array-type data, the data in the array can be of any type.

Return Value:

Returns the table generation function.

Example:

```
select posexplode(array('a','c','f','b')) from dual;
Results:
+------------+-----+
| pos | val |
+------------+-----+
| 0 | a |
| 1 | c |
| 2 | f |
| 3 | b |
```

```
+------------+-----+
```

STRUCT

Function definition:

```
STRUCT struct(value1,value2, ...)
```

Usage:

Creates a struct using the given value list.

Parameter description:

value: Each value can be of any type.

Return Value:

Returns the `STRUCT<col1:T1, col2:T2, ... >` Type. field names are sequential: col1, col2, …

Example:

```
select struct('a',123,'ture',56.90) from dual;
Results:
{col1:a, col2:123, col3:ture, col4:56.9}
```

NAMED_STRUCT

Function definition:

```
STRUCT named_struct(string name1, T1 value1, string name2, T2 value2
, ...)
```

Usage:

Creates a struct using the given name/value list.

Parameter description:

- value: Each value can be of any type.
- name: Specifies the name of a String-type field.

Return Value:

Returns the `STRUCT<name1:T1, name2:T2, ... >`type. The field names of the generated struct are sequential: name1, name2, …

Example:

```
select named_struct('user_id',10001,'user_name','LiLei','married','F
','weight',63.50) from dual;
```

```
Results:
{user_id:10001, user_name:LiLei, married:F, weight:63.5}
```

INLINE

Command Format:

```
inline(array<struct<f1:T1, f2:T2, ... >>)
```

as shown in the following figure:

Explodes the given struct array. Each element is given one row and each struct element corresponds to one column in each row.

Parameter description:

`STRUCT<f1:T1, f2:T2, ... >`: The values in the array can be of any type.

Return Value:

Returns the table generation function.

Example:

If the field in Table t_table is (`t_struct struct<user_id:bigint,user_name:string, married:string,weight:double>` <user_id: bigint,="" user_name:="" string,="" married:="" weight:="" double="">), the table data is as follows:</user_id:>

```
+----------+
| T_struct |
+----------+
{user_id:10001, user_name:LiLei, married:F, weight:63.5}
{user_id:10001, user_name:LiLei, married:F, weight:63.5}
+----------+
```

Execute SQL:

```
select inline(array(t_struct)) from t_table;
```

Return result:

```
+------------+----+----+-----------+-----------+
| user_id    | user_name | married | weight     |
+------------+----+----+-----------+-----------+
| 10001      | LiLei     | N       | 63.5       |
| 10002      | HanMeiMei | Y       | 43.5       |
```

```
+------------+-----------+---------+------------+
```

TRANS_ARRAY

Function definition:

```
trans_array (num_keys, separator, key1,key2,…,col1, col2,col3) as (
key1,key2,…,col1, col2)
```

Usage:

A UDTF that converts one row of data to multiple rows, and converts an array separated with fixed -separator format in column into multiple rows.

Parameter description:

- num_keys: Bigint type constant, must be larger than or equal to 0. It is used as the number of columns to transpose key when converting to multiple rows.
- Key: Duplicate columns in multiple rows when converting one row to multiple rows.
- separator: String type constant. It is a separator used to split a string into multiple elements. Exception is thrown when it is null.
- keys: As column of key when you transpose. It is specified by num_keys. If num_keys specifies that all columns are keys (that is, num_keys equals the number of all columns), only one row is returned.
- cols: An array to convert to rows. All columns after keys are considered as an array to be transposed. String type. The stored contents are arrays of string format, such as “Hangzhou; Beijing; shanghai”, they are arrays separated by “;”.

Return Value:

Transposed rows, new column names are specified by as. The type of column that is as key remains unchanged, and all other columns are String type. The number of rows to be split depends on the array that has maximum number, no-value locales are complemented with NULL.

Note:

The following restrictions apply when using UDTF:

- All columns that are considered as keys must be placed front, and columns to be transposed must be placed behind.
- One select can only have one UDTF and no other columns can appear.
- One select can only have one UDTF and no other columns can appear.

Example:

The data in the t_table table is as follows:

```
+----------+----------+------------+
| login_id | login_ip | login_time |
+----------+----------+------------+
wangwangA 192.168.0.1,192.168.0.2 20120101010000,20120102010000
| Wangwangb | 192.168.25.10, 192.168.67.22, 192,168.6. 3 | maid,
20120223080000 |
+----------+----------+------------+
```

Execute SQL:

```
trans_array(1, ",", login_id, login_ip, login_time) as (login_id,
login_ip,login_time)
```

Results:

```
+----------+----------+------------+
| 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 |
+----------+----------+------------+
```

If the table contains the following data:

```
Login_id LOGIN_IP LOGIN_TIME
wangwangA 192.168.0.1,192.168.0.2 20120101010000
```

NULL is complemented to the no-value locales in the array:

```
Login_id Login_ip Login_time
wangwangA 192.168.0.1 20120101010000
wangwangA 192.168.0.2 NULL
```

4.9 UDF

4.9.1 UDF Summary

MaxCompute provides many built-in functions to meet the computing requests of the users.

A User Defined Function (UDF) is similar to any other *Built-in Function*. For the corresponding relationship between Java and MaxCompute data types, see *Parameters and Return Value Types*.

If you use *Maven* to search “odps-sdk-udf” from *Maven* to get different versions of Java SDK,the configuration is as follows:

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
```

```
    <artifactId>odps-sdk-udf</artifactId>
   <version>0.29.10-public</version>
</dependency>
```

In MaxCompute, you can expand two types of UDF:

UDF Class	Description
UDF(User Defined Scalar Function)	User Defined Scalar function. The relationship between input and output is a one-to-one relationship. Read a row data and write an output value.
UDTF (UserDefined Table Valued Function)	User-defined table valued functions are used in scenarios where the calling of one function leads to multiple rows of data being output. It is a unique user-defined function which can return multiple fields, while UDFcan only output a return value.
UDAF (User Defined Aggregation Function)	User Defined Aggregation Function (UDAF), the relationship between its input and output is one-to-many relationships. That is to aggregate multiple input records to an output value. It can be used with a Group By clause.. For more information, see *###Aggregation Functions*.

Note:

- UDF stands for the set of user-defined functions, including User Defined Scalar Function, User Defined Aggregation Function and User Defined Table Valued Function. In a narrower sense, it represents user User Defined Scalar Function. The document uses this term frequently and the readers can judge the specific meaning according to the context .
- If the system prompts that memory is insufficient with an UDF involved in the SQL statement, configure `set odps.sql.udf.joiner.jvm.memory=xxxx;` to resolve this issue. This is because the data is huge and data skew also exists., This leads the memory size to occupythe task, which exceeds the default memory size.

MaxCompute UDF supports cross-project sharing. A UDF in project_b can be used in project_a . For more information, , see Authorization in Security Guide documentation. other_project: udf_in_other_project(arg0, arg1) as res from table_t;。

UDF Examples

Please see *UDF Example* in Quick Start Volume.

4.9.2 Java UDF

MaxCompute UDF includes three types: UDF, UDAF, and UDTF. This article focuses on how to implement these three functions through Java.

Parameter and return value type

The data types of UDF supported by MaxCompute SQL include thebasic types: bigint, double , boolean, datetime, decimal, string, tinyint, smallint, int, float, varchar, binary, and timestamp. Complex types: array, map, and struct.

- The use of some basic types including tinyint, smallint, int, float, varchar, binary, and timestamp through Java UDF is as follows:
  - UDTF get 'signature' by @Resolve annotation, for example, `@Resolve("smallint->varchar(10)")`.
  - UDF gets 'signature' by the reflection analysis 'evaluate'. In this case, the MaxCompute built-in type and the Java type comply with one-to-one mapping.
  - UDAF gets the signature with the @Resolve annotation, and maxcompute2.0 supports the use of new types in annotations, for example, `@Resolve("smallint-> varchar (10 )")`.
- JAVA UDF uses three complex data types :'array', 'map', and 'struct':
  - UDAFs and UDTFs specify signature by @Resolve annotation, for example, `@Resolve("array<string>,struct<a1:bigint,b1:string>,string->map<string,bigint>,struct<b1:bigint>")`.
  - The UDF maps the input and output types of the UDF through the signature of the evaluate method, reference is made to the mapping of the maxcompute type to the Java type. In this relationship, Array maps java.util.List, Map maps java.util.Map, and Struct maps com.aliyun.odps.data.Struct.
  - UDAF gets the signature with the @Resolve annotation, and MaxCompute2.0 supports the use of new types in annotations, for example, `@Resolve("smallint-> varchar (10 )")`.

Note:

- com.aliyun.odps.data.Struct does not see field name and field type from reflection, so it must be complemented by @Resolve annotation. In other words, to use Struct in a UDF , add the @Resolve annotation to the UDF class. This annotation only affects overloads of parameters or return values that contain com.aliyun.odps.data.Struct.

- *Currently, only one @Resolve annotation can be provided on class. Therefore, only one overload in a UDF with a struct parameter or return value can exist.*

The following table lists the relations 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
Struct	com.aliyun.odps.data.Struct

Note:

- The corresponding data type in Java and the return value data type is the object. Make sure that the first letter is uppercase.
- The NULL value in SQL is represented by a NULL reference in Java; therefore, ‘Java primitive type’ is not allowed because it cannot represent a NULL value in SQL.
- Here, Java type corresponding to the ‘array’ type is ‘list’.

UDF

To implement UDF, the class ‘com.aliyun.odps.udf.UDF’ must be inherited and the ‘evaluate’ method must be applied. The ‘evaluate’ method must be a non-static public method. The parameter type and return value type of Evaluate method is considered as UDF signature in

SQL. It means that the user can implement multiple evaluate methods in UDF. To call UDF, the framework must match the correct evaluate method according to the parameter type called by UDF.

Note：Classes with the same class name but different functional logic mustappear in different jar packages. For example, UDF (UDAF/UDTF): udf1, udf2 correspond to the resources udf1.jar and udf2.jar respectively, if both jars contain com.aliyun.UserFunction.class, when two udfs are used in the same SQL statement, the system randomly loads one of the classes. This causes inconsistency in the udf execution behavior or compilation failure.

UDF samples are as follows:

```
package org.alidata.odps.udf.examples;
  import com.aliyun.odps.udf.UDF;

public final class Lower extends UDF {
  Public String evaluate (string s ){
    If (Stream = NULL ){
        return null;
    }
        return s.toLowerCase();
  }
}
```

UDF is initialized and terminated through `void setup(ExecutionContext ctx)` and `void close()`.

The use method of UDF is similar to built-in functions in MaxCompute SQL. For more information, see *Built-in Functions*.

Other UDF examples

In the following code, UDF with three overloads is defined. The first, second, and third overloads use ARRAY, MAP, and STRUCT respectively as a parameter. Since the third overloads use a struct as a parameter or return value, therefore, a `@Resolve` annotation must be placed on the UDF class to specify the specific type of struct.

```
@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;
 }
```

```
}
```

The user can pass the complex type directly into the UDF:

```
create function my_index as 'UdfArray' using 'myjar.jar';
select id, my_index(array('red', 'yellow', 'green'), colorOrdinal) as
color_name from colors;
```

UDAF

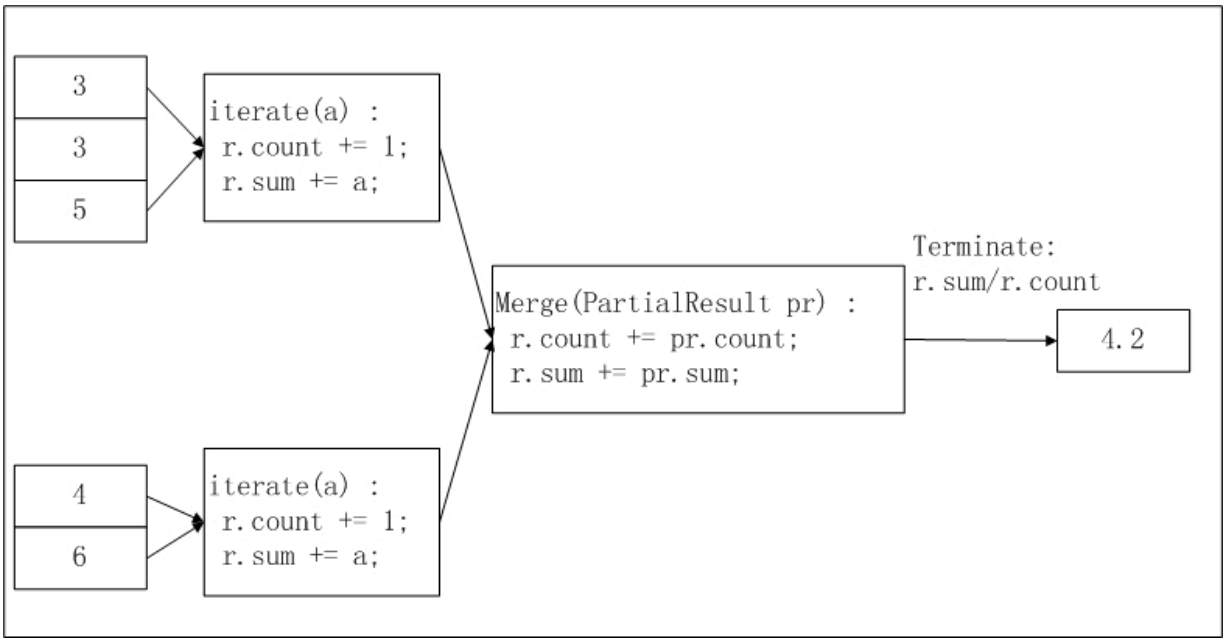
To implement Java UDAF, inherit the class ‘com.aliyun.odps.udf.Aggregator’ and the following interfaces must be applied:

```
public abstract class Aggregator implements ContextFunction {
  @Override
  public void setup(ExecutionContext ctx) throws UDFException {

  }
  @Override
  public void close() throws UDFException {
  }
  /**
   * Create an aggregate buffer
   * @return Writable - Aggregate buffer
   */
  abstract public Writable newBuffer();
  /**
   * @param buffer: aggregation buffer
   * @param args: specified parameter to call UDAF in SQL
   * @throws UDFException
   */
  abstract public void iterate(Writable buffer, Writable[] args)
throws UDFException;
  /**
   * generate final result
   * @param buffer
   * @return final result of Object UDAF
   * @throws UDFException
   */
  abstract public Writable terminate(Writable buffer) throws
UDFException;
  abstract public void merge(Writable buffer, Writable partial) throws
 UDFException;
}
```

The three most important interfaces are ‘iterate’, ‘merge’, and ‘terminate’. The main logic of UDAF relies on these three interfaces. In addition, user must realize defined Writable buffer.

Take ‘achieve average calculation’ as an example and next figure describes the realization logical and computational procedure of this function in MaxCompute UDAF:



In the preceding figure , the input data is sliced according to a certain size.For more information about slicing, see *MapReduce*). The size of each slice is suitable for a worker to complete in the specified time. This slice size must be configured manually by the user.

The calculation process of UDAF is divided into two steps:

- In the first step, 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.
- In the second step, a worker gathers the information of each slice generated in the first stage. In the final output, r.sum / r.count is the average of all input data.

Use the following UDAF encoding example to calculate the average:

```
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;
  }
}
```

Note:

- For Writable's readFields function, since the partial writable object can be reused, the same object readFields function is called multiple times. This function expects the entire object to be reset each time it is called. If the object contains a collection, it must be emptied.
- The use method of UDAF is similar to aggregation functions in MaxCompute SQL. For more information, see *Aggregation Functions*.
- How to run UDTF is similar to UDF. For more information, see *Java UDF Development*.

UDTF

Java UDTF class must inherit the class 'com.aliyun.odps.udf.UDTF'. This class has four interfaces:

Interface Definition	Description
public void setup(ExecutionContext ctx) throws UDFException	The initialization method to call user-defined initialization behavior before UDTF processes the input data. ‘Setup’ will be called first and once for each worker.
public void process(Object[] args) throws UDFException	The framework calls this method. Each record in SQL calls ‘process’ once accordingly. The parameters of ‘process’ are the specified UDTF input parameters in SQL. The input parameters are passed in as Object[], and the results are output through ‘forward’ function. The user must call ‘forward’ in the ‘process’ function by itself to determine the output data.
public void close() throws UDFException	The termination method of UDTF. The framework calls this method, and only once; that is, after processing the last record.
public void forward(Object …o) throws UDFException	The user calls the ‘forward’ method to output data. Each ‘forward’ represents the output of a record, corresponding to the column specified by UDTF 'as’ clause in SQL.

A UDTF program sample is as follows:

```
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 preceding example is for reference only. How to run UDTF is similar to using UDF. For more information, see *Java UDF Development*.

In SQL,use this UDTF as the following example. Suppose that the register function name in MaxCompute is ‘user_udtf’.

```
select user_udtf(col0, col1) as (c0, c1) from my_table;
```

Suppose the values of col0 and col1 in my_table are:

```
+------+------+
| col0 | col1 |
+------+------+
| A B | 1 |
| C D | 2 |
+------+------+
```

Then the ‘SELECT’ result is:

```
+----+----+
| c0 | c1 |
+----+----+
| A | 1 |
| B | 1 |
| C | 2 |
| D | 2 |
+----+----+
```

Instructions

UDTFs are often used as following in SQL:

```
select user_udtf(col0, col1) 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;
```

But using UDTF has the following limits:

- Other expressions are not allowed in the same SELECT clause:

  ```
  select value, user_udtf(key) as mycol ...
  ```

- UDTF cannot be nested.

  ```
  select user_udtf1(user_udtf2(key)) as mycol...
  ```

- It cannot be used with ‘group by / distribute by / sort by’ in the same SELECT clause.

  ```
  select user_udtf(key) as mycol ... group by mycol
  ```

Other UDTF Examples

In UDTF, learn more aboutMaxCompute *Resources*. The following describes how to use UDTFs to read MaxCompute resources:

1. Compile a UDTF program. Once the compilation is successful, export the Jar package (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_reso
urce1").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 in 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 UDTF (my_udtf) in MaxCompute:

```
create function mp_udtf as com.aliyun.odps.examples.udf.UDTFResource
 using
'udtfexample1.jar, file_resource.txt, table_resource1, table_reso
urce2';
```

4. Create the resource tables: table_resource1, table_resource2 and the physical table tmp1 in MaxCompute. Insert corresponding data into the tables.

5. Run this UDTF.

```
select mp_udtf("10","20") as (a, b, fileResourceLineCount) from tmp1
;
Return result:
+-------+-----------+-------+
| a | b | fileResourceLineCount |
+-------+-----------+-------+
| 10 | 2 | fileResourceLineCount=3|tableResource1RecordCount=0|
tableResource2RecordCount=0 |
| 10 | 2 | fileresourcelinecount = 3 | tableResource1RecordCount = 0
 | tableResource2RecordCount = 0 |
+-------+-----------+-------+
```

UDTF Examples—Complex Data Types

The code in the following example defines UDF with three overloads. The first overload uses ‘array’ as the parameter; the second uses ‘map’ as the parameter; and the third uses ‘struct’ as the parameter. Since the third overload uses ‘struct’ as the parameter or returned value, the UDF class must havethe @Resolve annotation to specify the specific type of ‘struct’.

```
@Resolve("struct<a:bigint>,string->string")
public class UdfArray extends UDF {
  public String evaluate(List<String> vals, Long len) {
    return vals.get(len.intValue());
  }
  public String evaluate(Map<String,String> map, String key) {
    return map.get(key);
  }
  public String evaluate(Struct struct, String key) {
    return struct.getFieldValue("a") + key;
  }
```

```
}
```

Users can pass in the complex data type in the UDF:

```
create function my_index as 'UdfArray' using 'myjar.jar';
select id, my_index(array('red', 'yellow', 'green'), colorOrdinal) as
color_name from colors;
```

Hive UDF Compatibility Example

MaxCompute 2.0 supports Hive-style UDFs. Some Hive UDFs and UDTFs can be used directly in MaxCompute.

Notice:

Currently, the compatible Hive version is 2.1.0, and the corresponding Hadoop version is 2.7.2. UDFs that are developed in other versions of Hive/Hadoop may need to be recompiled using this Hive/Hadoop version.

Example:

```
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.ObjectInsp
ectorFactory;
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:

For the use of Hive UDF, see:

- *https://cwiki.apache.org/confluence/display/Hive/HivePlugins*
- *https://cwiki.apache.org/confluence/display/Hive/DeveloperGuide+UDTF*
- *https://cwiki.apache.org/confluence/display/Hive/GenericUDAFCaseStudy*

The UDF can pack any type and amount of parameters into array to output. Suppose that the output jar package is named test.jar:

```
--Add resource
Add jar test.jar;
--Create function
CREATE FUNCTION hive_collect as 'com.aliyun.odps.compiler.hive.Collect
' using 'test.jar';
--Use function
set odps.sql.hive.compatible=true;
select hive_collect(4y,5y,6y) from dual;
+------+
| _c0 |
+------+
| [4, 5, 6] |
+------+
```

Note:

The UDF supports all data types, including array, map, struct, and other complex types.

Note:

- MaxCompute's add jar command permanently creates a resource in the project, specify the jar when creating an UDF, but you cannot automatically add all jars to the classpath.
- To use compatible Hive UDF, add `set odps.sql.hive.compatible=true;` opposite the SQL statement, and submit it with SQL statement.
- When using compatible Hive UDFs, you must pay attention to *JAVA sandbox* limits of MaxCompute.

4.9.3 Python UDF

The MaxCompute UDF consists of UDF, UDAF, and UDTF functions. This article explains how to implement these three functions through MaxCompute Python.

RESTRICTED ENVIRONMENT

The Python version of MaxCompute UDF is 2.7 and executes user code in sandbox mode; that is, the code is executed in a restricted environment.

- Read and Write local files
- Promoter process
- Start thread
- Use SOCKET to communicate
- Other system calls

Because of these restrictions, user-uploaded code must be implemented throughj pure Python, and the C extension module is disabled.

In addition, not all modules are available in the Python standard library, and modules that involve these features are disabled. Description of available modules in the standard library are as follows:

- All modules implemented by pure Python are available.
- The following modules are available in C-implemented extended modules.
  - array
  - audioop
  - binascii
  - _bisect
  - cmath
  - _codecs_cn
  - _codecs_hk
  - _codecs_iso2022
  - _codecs_jp
  - _codecs_kr
  - _codecs_tw
  - _collections
  - cStringIO
  - datetime
  - _functools
  - future_builtins
  - _hashlib

  - _heapq
  - itertools
  - _json
  - _locale
  - _lsprof
  - math
  - _md5
  - _multibytecodec
  - operator
  - _random
  - _sha256
  - _sha512
  - _sha
  - _struct
  - strop
  - time
  - unicodedat
  - _weakref
  - cPickle
- Some modules have limited functionalities. For example, the sandbox limits the degree to which user code can write data to the standard output and the standard error output; that is, `sys.stdout/sys.stderr` can write 20 KB at most; otherwise, the excessive characters will be ignored.

Third-party Libraries

Common third-party libraries are installed in the operating environment to supplement the standard library. The supported third-party libraries also include **numpy**.

Note:

The use of third-party libraries is also subject to 'prohibit local', 'network I/O', and other restrictions . Therefore, APIs that have such functions are also prohibited in a third-party library.

Parameters and return value types

The parameters and return values are specified as follows:

```
@odps.udf.annotate(signature)
```

MaxCompute SQL data types that are currently supported by the Python UDF include bigint, String, double, Boolean, and datetime. The SQL statement must determine the parameter type and the return value type for all functions before execution. So for Python, a dynamically-typed language, you must specify the function signature by adding a decorator to the UDF class.

The function signature is specified by a string. The syntax is as follows:

```
arg_type_list '->' type_list
 arg_type_list: type_list | '*' | ''
type_list: [type_list ','] type
'bigint' | 'string' | 'double' | 'boolean' | 'datetime'
```

- The left side of the arrow indicates the type of the parameter and the right side indicates the type of the returned value.
- Only the UDTF returned value can be multiple columns, while UDF and UDAF can only return one column.
- '*' represents varargs. By using varargs, UDF/UDTF/UDAF can match any type of parameter.

A valid signature example is as follows:

```
The 'bigint, double-> string' # parameter is bigint, double, and the
return value is string

The 'bigint, boolean-> string, datetime '# udtf parameter is bigint,
Boolean, the return value is string, datetime

'*->string' #  variable length parameter, input parameter arbitrary,
return value string

The '-> doubles' # parameter is empty and the return value is double
```

At the query semantic parsing stage, unqualified signatures are removed, and an error is returned. The execution is then stopped. During execution, the UDF parameter will be passed to the user as the type specified by the function signature. The type of the user returned value must be consistent with the type specified by the function signature; otherwise, an error is returned. MaxCompute SQL data type corresponds to the Python type as follows:

ODPS SQL type	Bigint	String	Double	Boolean	Datetime
Python Type	int	str	float	bool	int

Note:

- Datetime type is passed to user code in the form of an int, with a value of epoch UTC Number of milliseconds from time to date. The user can deal with 'datetime' type through the 'datetime' module in the Python standard library.
- NULL corresponds to NONE in Python.

In addition, the parameter of odps.udf.int(value[, silent=True]) has been adjusted. Parameter 'silent' is added. . When 'silent' is true, if the value cannot be converted into 'int', report no error and return NONE.

UDF

Implementation of the Python UDF is very simple. You are required to define a **`new-style class`**, and implements the **`evaluate`** method. For example:

```
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 through annotate.

Since October 16, 2018, the use of Python UDF in the MaxCompute public cloud environment has been fully opened.

UDAF

- **`class odps.udf.BaseUDAF`**: Inherit this class to implement a Python UDAF.
- BaseUDAF.new_buffer(): Implement this method and return the median 'buffer' of the aggregate function. Buffer must be *marshallable* object (such as list, dict), and the size of the buffer must not increase with the amount of data, in case of limit, Buffer size after Marshal must not exceed 2 MB.
- BaseUDAF.iterate(buffer[, args, ...]): This method aggregates 'args' into the median 'buffer'.
- BaseUDAF.merge(buffer, pbuffer): This method aggregates two median buffers; that is, aggregate 'pbuffer merger' into 'buffer'.

- BaseUDAF.terminate(buffer): This method converts the median ‘buffer’ into the MaxCompute SQL basic types.

An example of an average value of UDAF is as follows:

```
@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]
```

UDTF

- **`class odps.udf.BaseUDTF`**: The basic class of Python UDTF. Users inherit this class and implement methods such as process, close, and so on.
- BaseUDTF.__init__(): The initialization method, the inheritance class, if you implement this method, the base class's initialization method, super(BaseUDTF, self).__init__() must be called in the beginning.

  The **`init`** method can only be called once during the entire UDTF life cycle; that is, before the first record is processed. When the UDTF must save the internal state, all states can be initialized in this method.
- BaseUDTF. process ([args,...]): This is one of the MaxCompute methods. The framework calls this method. Each record in SQL calls ‘process’ once accordingly. The parameters of ‘process’ are the specified UDTF input parameters in SQL.
- BaseUDTF.forward([args, ...]): The UDTF output method, which is called by user codes. Each time ‘forward’ is called, a record is output. The parameters of ‘forward’ are the UDTF output parameters specified in SQL.
- BaseUDTF.close(): The termination method of UDTF. This method is called by the MaxCompute SQL framework and only to be called once; that is, after processing the last record.

Examples of UDTF are:

```
#coding:utf-8
# explode. py


from odps.udf import annotate

from odps.udf import BaseUDTF



@annotate('string -> string')
class Explode(BaseUDTF):
   """Output string comma-separated to multiple records
   """

   def process(self, arg):
       props = arg.split(',')
        for p in props:
           self.forward(p)
```

Note:

A Python UDTF can also specify the parameter type or returned value type without adding ‘annotate’. In this case, the function can match any input parameter in SQL. The returned value type cannot be deduced, but all output parameters will be considered to be ‘String’ type. So when ‘forward’ is called, all output values must be converted into ‘str’ type.

Referring to resources

Python UDF can reference resource files through the ‘odps.distcache’ module. Currently, referencing file resources and table resources are supported.

- odps.distcache.get_cache_file(resource_name)
  - Returns the resource content for the specified name. resource_name: ‘str’ type, correspond ing to the existing resource name in the current project. If the resource name is invalid or has no responding resources, returns an error.
  - The return value is file-like object the caller must call the **`close`** method to release the open resource file after this object has been used.

  The example of using ‘get_cache_file’ is as follows:

```
@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 contents of the specified resource table. resource_name: ‘str’ type, corresponding to the existing resource table name in the current project. If the resource name is invalid or has no responding resources, returns an error.
  - Returned value: Returned value is a ‘generator’ type. The caller obtains the table content through traversal. Each traversal has a record stored in the table in the form of a tuple.

  The example of using ‘get_cache_table’ is as follows:

```
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)
```

4.10 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

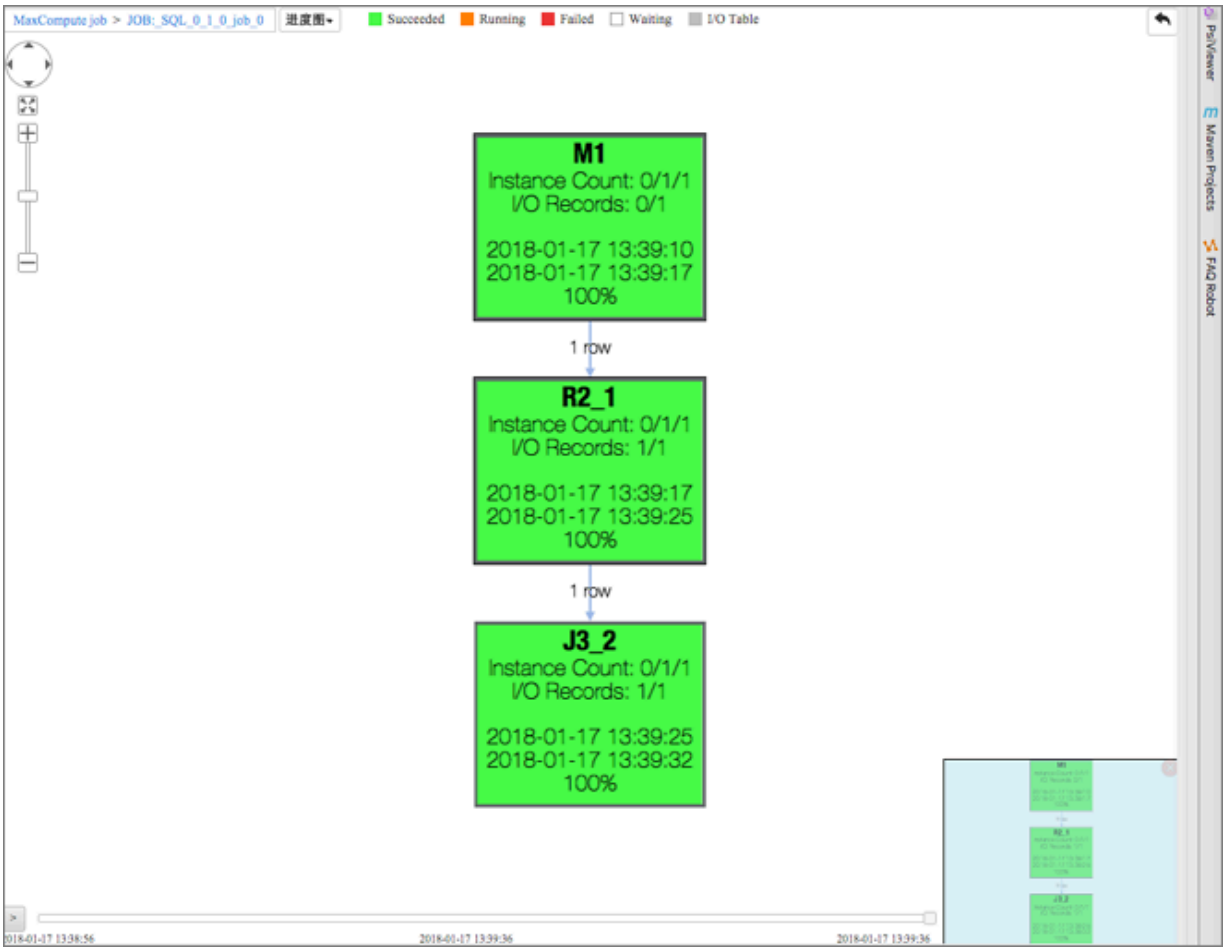
The example shows how to use UDT to access Java static fields. You can use UDT to implement more functions. The following example shows the UDT execution procedure and its functions.

```
--  Sample data
@table1 := select * from values ('100000000000000000000') 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:

```
100000000000000000100
```

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 protobuffer. 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:

  1. To input multiple rows or columns, reference the example of using the aggregate function.
  2. 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.
  3. 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 Object 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.

4.11 UDJ

Based on the MaxCompute 2.0 computing engine, MaxCompute introduces a new interface: user defined join (UDJ) to the user defined function (UDF) framework. This interface allows you to handle multiple tables more flexibly based on user-defined methods. It also simplifies the operations performed in the underlying distributed system using MapReduce. This is a major improvement of MaxCompute in big data processing based on NewSQL.

Overview

MaxCompute provides multiple native *Join* methods, including INNER JOIN, RIGHT JOIN, OUTER JOIN, LEFT JOIN, FULL JOIN, SEMIJOIN , and ANTISEMIJOIN methods. You can use these native join methods in most scenarios. However, these methods do not support handling multiple tables.

In most cases, you can build your code framework using UDFs. However, the current UDF, UDTF , and UDAF frameworks only support handling one table. To perform user-defined operations for multiple tables, you have to use native join methods, UDFs, UDTFs, and complex SQL statements . In certain cases, when handling multiple tables, you even have to use custom MapReduce framework instead of SQL, in order to complete the required computing task.

In any situation, these operations require technological expertise and may cause the following issues:

- For the computing platform, calling multiple join methods in SQL statements may cause a " black box," which is complex and difficult to execute with the least overheads.
- Using MapReduce even make optimal execution of code becomes impossible. Most of the MapReduce code is written in Java. The execution of the MapReduce code is less efficient than the execution of MaxCompute code generated by the LLVM code generator at an optimized native runtime.

Examples

The following example describes how to use UDJ in MaxCompute. This example uses the payment table and the user_client_log table.

- The payment (user_id string,time datetime,pay_info string) table stores the payment informatio n of a user. Each payment record includes the user ID, payment time, and the payment details.
- The user_client_log(user_id string,time datetime,content string) table stores the client log records for a user. Each record contains the user ID, operation time, and operation.

Requirement: For each record in the user_client_log table, locate the payment record that has the time closest to the operation time, and join and output the content of both records.

The sample data is as follows:

payment

user_id	time	pay_info
2656199	2018-02-13 22:30:00	gZhvdySOQb

user_id	time	pay_info
8881237	2018-02-13 08:30:00	pYvotuLDIT
8881237	2018-02-13 10:32:00	KBuMzRpsko

user_client_log

user_id	time	content
8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn
8881237	2018-02-13 06:14:00	click OkTYNUHMqZzlDyL
8881237	2018-02-13 10:30:00	click OkTYNUHMqZzlDyL

The following is a record in the user_client_log table.

user_id	time	content
8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn

The following payment record has the time closest to the operation time in the preceding client log record.

user_id	time	pay_info
8881237	2018-02-13 08:30:00	pYvotuLDIT

These two records are merged as follows:

8881237	2018-02-13 00:30:00	click MpkvilgWSmhUuPn, pay pYvotuLDIT

The merging result of the two tables is as follows:

user_id	time	content
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

Call native join operations

If you use standard join methods, you have to join these two tables based on the common user_id field. You must locate the payment record that has the closest time to the operation time in each client log record. The SQL statement may be written 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 rows in the tables, you must calculate the minimum difference between the p.time and u.time under the same user_id. However, you cannot call the aggregate function in the join condition. Therefore, this task cannot be completed by calling the standard join method.

In a distributed system, the join method merges rows retrieved from two or more tables based on a field that is shared by these tables. If you use the join method in standard SQL, you only have a few options to handle the merged data. Therefore, a generic interface, such as UDJ, is required to handle the merged data in a customized manner and output the result. This interface may be integrated as a plug-in.

Use Java to write UDJ code

The following describes how to use UDJ to achieve a function that cannot be implemented by calling a native join method.

Prerequisites

Since UDJ is a new feature, a new SDK is required.

```
<dependency>
  <groupId>com.aliyun.odps</groupId>
  <artifactId>odps-sdk-udf</artifactId>
  <version>0.29.10-public</version>
  <scope>provided</scope>
</dependency>
```

The SDK contains a new abstract class UDJ. By extending this UDJ, you can implement all UDJ functions.

```
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 of the right table, find the nearest record of the
 left table and
 * merge two records.
 */
@Resolve("->string,bigint,string")
public class PayUserLogMergeJoin extends UDJ {
  private Record outputRecord;
  /** Will be called prior to the data processing phase. User could
implement
   * use this method to do initialization work.
   */
  @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)
    });
  }
  /** Override this method to implement join logic.
   * @param key Current join key
   * @param left Group of records of the left table corresponding to
the current key
   * @param right Group of records of the right table corresponding to
 the current key
   * @param output Used to output the result 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()) {
      // Empty the right group and do nothing.
      return;
    } else if (! left.hasNext()) {
      // Empty left group. Output all records of the right group
without merge.
      while (iter.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 left group of records will be iterated from the start to
the end
    // for each record of the right group, but the iterator cannot be
reset.
    // So we save every record of the left to an 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 through all records of the left, and find the pay
record that has
      // the minimal difference in terms of time.
      for (Record pay: pays) {
        long delta = Math.abs(logTime - pay.getDatetime(0).getTime());
        if (delta < minDelta) {
          minDelta = delta;
          nearestPay = pay;
        }
      }
      // Merge the log record with the nearest pay record and output
the result.
      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() {
  }
}
```

Note:

In this example, the NULL values in the entries are not processed. To simplify the data processing procedure for better presentation, this example assumes that no NULL values are contained in the tables.

Each time you call this join method of UDJ, records that match the same key in the two tables are returned. Therefore, UDJ searches all records in the payment table to locate the record with the time closest to each record the user_client_log table.

Assume that the user only has a few payment records. In this case, you can load the data in the payment table to the RAM. Typically, the RAM has enough space to store the payment data of a user generated in one day. What if this assumption is invalid? How can we resolve this issue? This issue will be discussed in topic Use the SORT BY clause

Create a UDJ in MaxCompute

After you have written the UDJ code in Java, upload the code to MaxCompute SQL as a plug-in. You must register the code with MaxCompute first. Assume that the code is packed into JAR package *odps-udj-example.jar*.

Use the Add JAR command to upload the JAR package to MaxCompute as follows:

```
add jar odps-udj-example.jar;
```

Use the CREATE FUNCTION statement to create UDJ function pay_user_log_merge_join, referencing the corresponding Java class and using the odps-udj-example.jar JAR package.

`com.aliyun.odps.udf.example.udj.PayUserLogMergeJoin:`

```
create function pay_user_log_merge_join
  as 'com.aliyun.odps.udf.example.udj.PayUserLogMergeJoin'
  using 'odps-udj-example.jar';
```

Use UDJ in MaxCompute SQL

After you have registered UDJ in the database, the function can be used by MaxCompute SQL.

1. Create a sample source table

```
create table payment (user_id string,time datetime,pay_info string);
create table user_client_log(user_id string,time datetime,content
string);
```

2. Create sample data:

```
Create the data in 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')
;
--Create data in the user_client_log table
INSERT OVERWRITE TABLE user_client_log VALUES
('1000235', datetime '2018-02-13 00:25:36', 'click FNOXAibRjkIaQPB
'),
('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 gbAfPCwrGXvEjpI
'),
('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 XIhZdLaOocQRmrY
'),
```

```
('4356142', datetime '2018-02-13 18:30:00', 'click DYqShmGbIoWKier
'),
('4356142', datetime '2018-02-13 19:54:00', 'click DYqShmGbIoWKier
'),
('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')
;
```

3. In MaxCompute SQL, use the UDJ function you have created:

```
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 syntax of UDJ is similar to the standard join syntax. The only difference is that the USING clause is added to UDJ.

- The name of the UDJ function in SQL is pay_user_log_merge_join.
- (p.time, p.pay_info, u.time, u.content) are the columns used in these two tables.
- r is the alias of the result returned by the UDJ function. You can reference this alias in other SQL statements.
- (user_id, time, content) are the columns returned by the UDJ function.

Execute this SQL statement, and the result is as follows:

```
+---------+------------+---------+
| user_id | time       | content |
+---------+------------+---------+
| 1000235 | 2018-02-13 00:25:36 | click FNOXAibRjkIaQPB |
| 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 nYHJqIpjevkKToy, pay
pYvotuLDIT |
| 2656199 | 2018-02-13 21:05:00 | click gbAfPCwrGXvEjpI, 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 XIhZdLaOocQRmrY, pay
gZhvdySOQb |
| 4356142 | 2018-02-13 18:30:00 | click DYqShmGbIoWKier |
| 4356142 | 2018-02-13 19:54:00 | click DYqShmGbIoWKier |
| 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 |
+---------+------------+---------+
```

As shown in the preceding code, the task that could not be performed by calling native join methods has been completed by using UDJ.

Pre-sorting

To locate the matching payment record, an iterator is used to search all 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 applied when the number of payment records is small. If a large number of payment records has been generated, due to RAM size limits, you must find another method to load the data. This section describes how to address this issue using the SORT BY clause.

When the size of the payment data is too large to be stored in the RAM, it would be easier to address this issue if all data in the table has already been sorted by time. You then only need to compare the first element in these two lists.

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 delta of time between two records is decreasing, we can
still
      // explore the left group to try to gain a smaller delta.
      lastPayRecord = payRecord.clone();
      prevDelta = delta;
      payRecord = left.next();
    } else {
      // Hit to the point of minimal delta. Check with the last pay
record,
      // output the merge result and prepare to process the next
record of
      // right group.
      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;
      }
    }
  }
}
```

In the native SQL language, you only need to make a few modifications to this example and add a SORT BY clause to the end of the UDJ clause, and then sort the data in both tables by time. Note : After you have modified the UDJ code, you must update the corresponding JAR package.

```
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)
SORT BY p.time, u.time
;
```

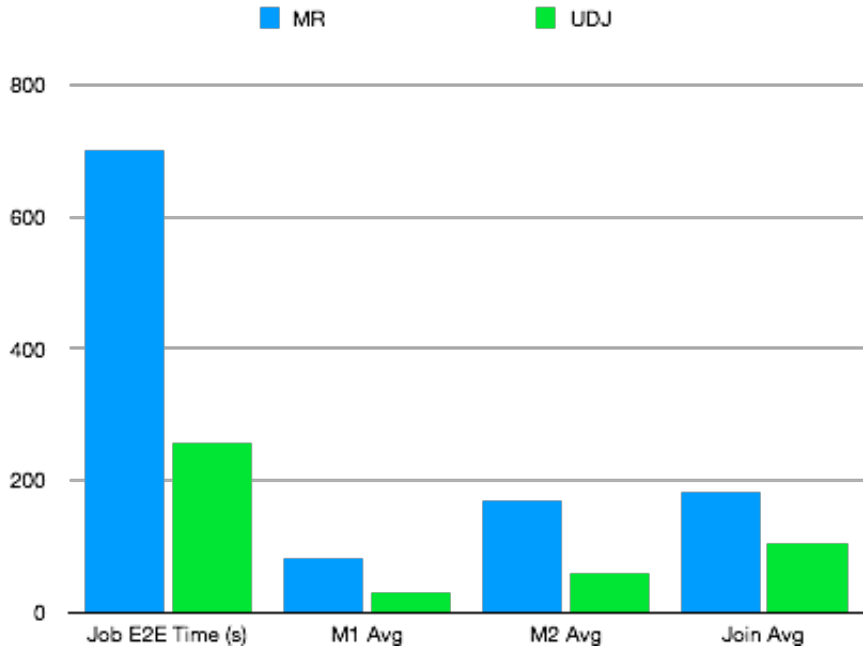
The execution result is the same as the result before the code is modified.

This method uses the SORT BY clause to pre-sort the data. To achieve the same result, only a maximum of three records need to be cached.

UDJ performance

Without UDJ, you have to use MapReduce to handle complex cross-table computing tasks in a distributed system. The applicable scenarios include complex business scenarios such as advertising and information search.

The following example uses an online MapReduce job to test the UDJ performance. This MapReduce job uses a complex algorithm to join two tables. This example uses UDJ to rewrite the SQL statements of the MapReduce job and then check whether the execution result is correct. Under the same programming concurrency, the comparison of performance is as follows:



As shown in the figure, using UDJ helps to describe the complex logic of handling multiple tables, and greatly improves the query performance. The code is only executed inside the UDJ function, and the entire logic of the code, such as the logic of the map stage in this example, is executed by the high-performance MaxCompute native runtime. UDJ optimizes the MaxCompute runtime engine and the data exchange between interfaces. The join logic of UDJ is more efficient than that at the reduce stage in MapReduce.

4.12 Differences with other SQL syntax

This article takes a SQL perspective. and introduces MaxCompute by comparing MaxCompute SQL with Hive, MySQL, Oracle, SQL Server Unsupported pant, and DML syntax.

Pant syntax not supported by MaxCompute

Grammar	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	Y	Y	N	Y	Y
Create Table-External table	Y (supports OSS and OTS External tables)	Y	N	N	N
Create Table-maid table	N	Y	Y	Y	Y (with # prefix)
Create Index	N	Y	Y	Y	Y
Virtual Column	N	N (only 2 predefined)	N	Y	Y

DML syntax not supported by MaxCompute

Grammar	MaxCompute	Hive	MySQL	Oracle	SQL Server
Select-recurrent CTE	N	N	N	Y	Y
Select-group by roll up	N	Y	Y	Y	Y
Select-group by cube	N	Y	N	Y	Y
Select-grouping set	N	Y	N	Y	Y
Maid join	Y	Y	N	Y	Y
Select-Fig	N	N	N	Y	Y
Select-correlated subquery	N	Y	Y	Y	Y
Set operator-Union (distinct)	Y	Y	Y	Y	Y
Set operator-intersect	N	N	N	Y	Y

Grammar	MaxCompute	Hive	MySQL	Oracle	SQL Server
Set operator-minus	N	N	N	Y	Y (keyword)
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	N	Y	N	N	N
Analytic-range	N	Y	N	Y	Y

4.13 SQL limits

Some users may fail to notice specific limits and find the service has stopped. The limits for MaxCompute SQL include the following:

Boundary name	Maximum value/ Limit	Class	Description
Length of table name	128 bytes	Length limit	Table names and column names cannot contain special characters. It must start with a letter and can contain only English letters (a-z, A-Z), numbers, and underscores (_).
Annotation length	1,024 bytes	Length limit	The annotation can contain valid strings for up to 1,024 bytes.
Column definitions	1,200	Quantity limit	One table can contain a maximum of 1,200 column definitions.
Partitions	60,000	Quantity limit	One table can contain a maximum of 60,000 partitions.
Partition levels of a table	6 levels	Quantity limit	A table can contain a maximum of six levels of partition.
Statistical definitions	100	Quantity limit	One table can contain a maximum of 100 statistical definitions.

Boundary name	Maximum value/ Limit	Class	Description
Statistical definitions	64,000	Length limit	A statistical definition can contain a maximum of 64,000 bytes.
Screen display	10,000 rows	Quantity limit	The screen display of a SELECT statement outputs a maximum of 10,000 rows.
INSERT targets	256	Quantity limit	A multiins operation can insert a maximum of 256 targets at a time.
UNION ALL	256	Quantity limit	The UNION ALL operation can be performed on a maximum of 256 tables.
MAPJOIN	Eight small tables	Quantity limit	A MAPJOIN operation can be performed on a maximum of eight small tables.
MAPJOIN memory restriction	512 MB	Quantity limit	The memory size of all small tables on which MAPJOIN operation is performed cannot exceed 512 MB.
Window functions	Five	Quantity limit	A SELECT statement can contain a maximum of five window functions.
ptinsubq	1,000 rows	Quantity limit	The results returned by PT IN SUBQUERY cannot exceed 1, 000 rows.
SQL statement	2 MB	Length limit	The maximum length of an SQL statement is 2 MB.
Number of conditions for a where clause	256	Quantity limit	A where clause can use a maximum of 256 conditions.
Length of column records	8 MB	Quantity limit	The maximum length of a cell in tables is 8 MB.

Boundary name	Maximum value/ Limit	Class	Description
Number of parameters of an in statement	1,024	Quantity limit	Specifies the maximum number of parameters of an in statement, for example, in (1,2,3….,1024). An excess of parameters of in(…) results in compilation pressure. 1,024 is a recommended value, not a limit value.
jobconf.json	1 MB	Length limit	The size of 'jobconf.json' is 1 MB . Including too many partitions in a table may cause 'jobconf.json' to exceed 1 MB.
View	Not writable	Operation restriction	A view cannot be written or operated using an insert statement.
Column data type	Not allowed	Operation limit	The data type and position of a column cannot be modified.
java udf function	Cannot be abstract or static	Operation limit	A Java UDF cannot be abstract or static.
A maximum of 10, 000 partitions can be queried.	10,000	Quantity limit	A maximum of 10,000 partitions can be queried.

Note:
The limits of MaxCompute SQL cannot be manually modified or configured.

4.14 Appendix

4.14.1 Escape characters

In MaxCompute SQL, a string constant can be set off by single (') or double quotation marks (").
The string set off by single quotation marks can contain double quotation marks or the string set off by double quotation marks can contain single quotation marks. Otherwise, you must use an escape character to indicate it.

The following expressions are acceptable:

```
"I'm a happy manong."
```

```
'I\'m a happy manong.'
```

In MaxCompute SQL, '\' is a kind of escape character used to express the special character in a string or express its followed characters as characters themselves. To read a string constant, if '\' is followed by three effective 8 hexadecimal digits and corresponding range is from 001 to 177, the system converts it to corresponding characters according to an ASCII value.

The following table lists some special escape characters:

Escape	Character
\b	backspace
\t	tab
\n	newline
\r	carriage-return
\'	single quotation mark
\"	double quotation marks
\ \	Backslash
\;	Semicolon
\Z	control-Z
\0 or \00	Terminator

```
select length('a\tb') from dual;
```

The result is 3, which indicates that three characters are in the string. The '\t' is considered as one character. Other following characters are expressed as themselves.

```
select 'a\ab',length('a\ab') from dual;
```

The result: 'aab', 3. '\a' is expressed as general 'a'.

4.14.2 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.

4.14.3 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]	^ indicats ‘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
[[:alnum:]]	letter and digit characters	[a-zA-Z0-9]
[[:alpha:]]	letter	[a-zA-Z]
[[:ascii:]]	ASCII character	[\x00-\x7F]
[[:blank:]]	Space character and tabs	[\t]
[[:cntrl:]]	Control character	[\x00-\x1F\x7F]
[[:digit:]]	Digit character	[0-9]
[[:graph:]]	Characters except white space characters	[\x21-\x7E]
[[:lower:]]	Lowercase characters	[a-z]
[[:print:]]	[:graph:] and white space characters	[\x20-\x7E]
[[:punct:]]	punctuation	[][!” [][!”#$%&’()*+,./:;<=>? @\^_ `{|}~-]
[[:space:]]	White space characters	[\t\r\n\v\f]
[[:upper:]]	Uppercase characters	[A-Z]
[[:xdigit:]]	hexadecimal character	[A-Fa-f0-9]

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 |
```

4.14.4 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
    ON OPTION OR ORDER OUT OUTER OUTPUTDRIVER
    OUTPUTFORMAT OVER OVERWRITE PARTITION PARTITIONED PARTITIONP
ROPERTIES 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 STORED
    STREAMTABLE STRING STRUCT TABLE TABLES TABLESAMPLE TBLPROPERTIES
    TEMPORARY TERMINATED TEXTFILE THEN TIMESTAMP TINYINT TO
    TOUCH TRANSFORM TRIGGER TRUE UNARCHIVE UNBOUNDED UNDO
    UNION UNIONTYPE UNIQUEJOIN UNLOCK UNSIGNED UPDATE USE
```

```
    USING UTC UTC_TMESTAMP VIEW WHEN WHERE WHILE DIV
```

4.14.5 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
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.

5 MapReduce

5.1 Summary

5.1.1 MapReduce

This article 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, SVM, and so on..
- 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

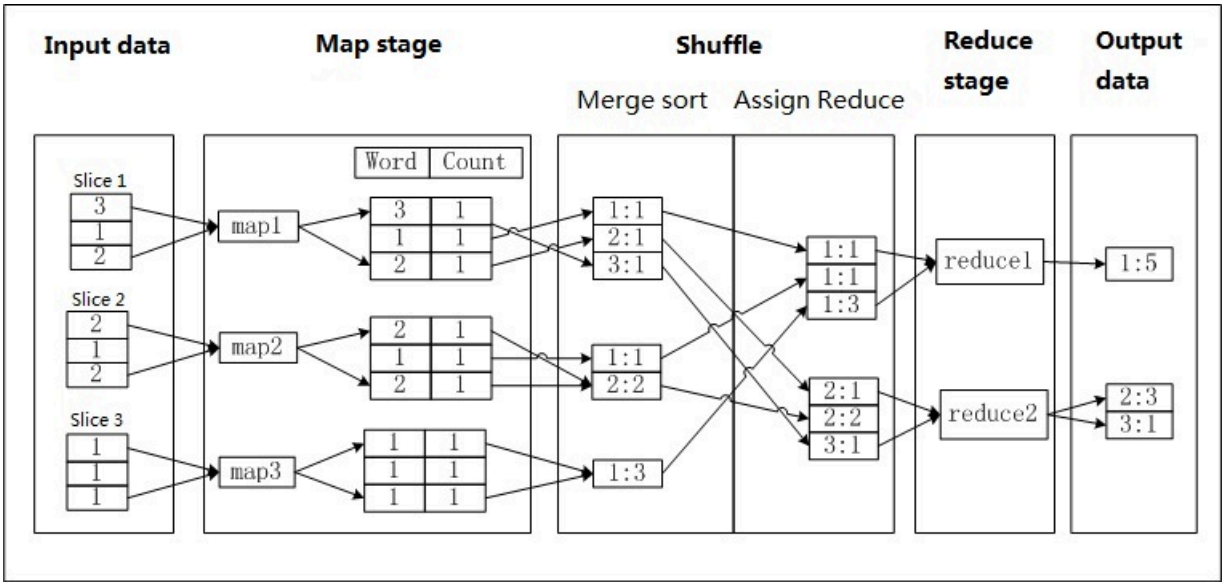
operation for multiple records of the same key. Then these multiple records become a value through Reduce processing.

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.

Assumethat 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.

5.1.2 Extended MapReduce

Compared with the traditional MapRudece, the extended MapReduce model provided by MaxCompute changes the underlying scheduling and I/O model, and avoids redundant I/O operations during the performance.

The traditional MapReduce model requires that the data must be loaded to the distributed file system (such as HDFS or MaxCompute table) after each round of MapReduce operation. However, a general MapReduce application usually consists of multiple MapReduce jobs, and each job output must be written to the disk. The following Map task is an example of a task used only to read the data, prepared for the subsequent Shuffle stage, but which actually results in redundant I/O operations.

The calculation scheduling logic of MaxCompute supports more complex programming paradigm. In the preceding scenario, the next Reduce operation can be executed after the Reduce operation and inserting a Map operation is not necessary. In this way, MaxCompute provides an extensiona l MapReduce model, that is, numerous Reduce operations can follow a Map operation, such as Map>Reduce> Reduce.

Hadoop Chain Mapper/Reducer also supports analogous serial Map or Reduce operations, but has major differences compared with the extensional MaxCompute (MR2) model.

The Hadoop Chain Mapper/Reducer is based on the traditional MapReduce model, and can only add one or multiple Mapper operations (it is not allowed to add Reducer operations) after the original Mapper or Reducer. The benefits of extended MapReduce are, a user can reuse previous business logic of Mapper and can split one Map stage or Reduce stage into multiple Mapper stages. The underlying scheduling and I/O model are not changed essentially.

Compared with *MaxCompute* , MR2 is basically consistent in a way Map/Reduce functions are written. The main difference is in the performance. For more information, see *Extended MapReduce example*.

5.1.3 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 *here* 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 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:

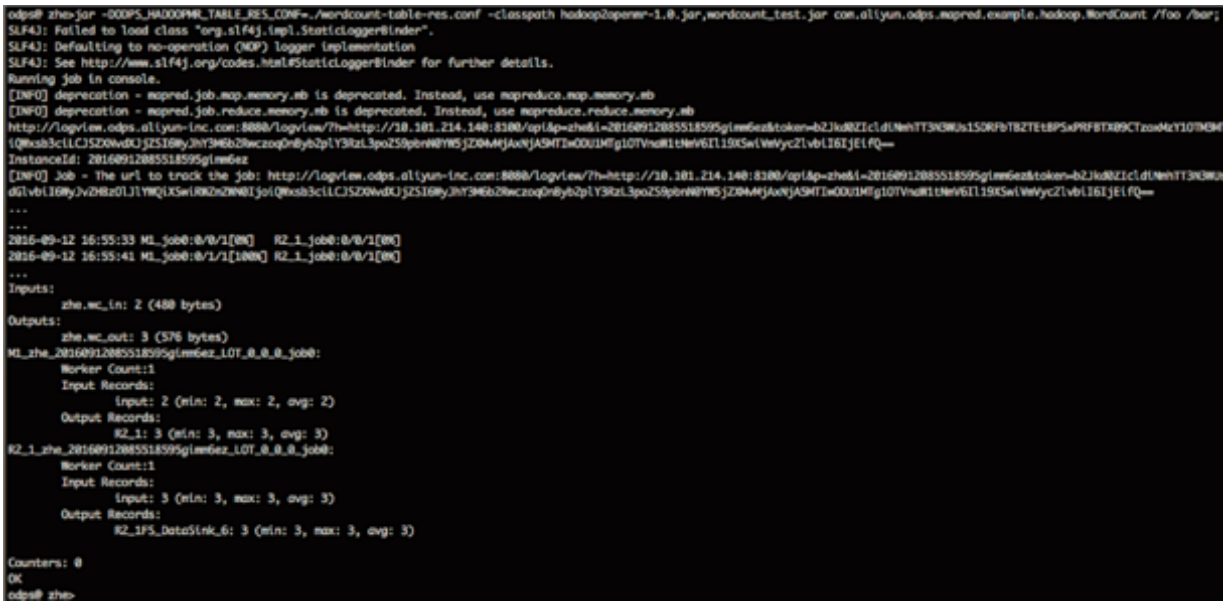
```
jar -DODPS_HADOOPMR_TABLE_RES_CONF=./wordcount-table-res.conf -
classpath hadoop2openmr-1.0.jar,wordcount_test.jar com.aliyun.odps.
mapred.example.hadoop.WordCount /foo/bar;
```

Note:

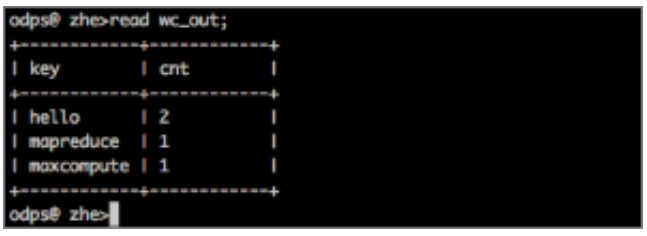
- wordcount-table-res.conf is a map with /foo/bar configured.
- wordcount_test.jar is your Jar package of Hadoop MapReduce.

- 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:



5.2 Function Introduction

5.2.1 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*.

5.2.2 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 configurat
ion file
        -resources <resource_name_list> file\table resources used in
mapper or reducer, seperate 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/resfile`.

  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;
```

5.2.3 Input and Output

- Built-in data types include: BIGINT, DOUBLE, STRING, DATETIME, and BOOLEAN. User-defined types (UDFs) are not supported.
- Multiple-table input is allowed, and the schema of input tables can be different. In a Map function, users can obtain corresponding Table information of the current record.
- The input can be null. View as an input is not supported.

- Reduce accepts multiple outputs and can output data to different tables or different partitions in the same table. The schema of different outputs can be different. Different outputs are distinguished through the label however, the default output does not need any label. An output cannot be empty.

For more input and output examples, see *Program Examples*.

5.2.4 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*.

5.2.5 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. ‘wcin’ 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.

Note:

The function to download the data from MaxCompute is only supported in MapReduce local operation mode. If the local debugging is executed in *Eclipse development plug-in*, the data of MaxCompute cannot be downloaded to local.

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.

5.3 Program Example

5.3.1 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 );
      }
```

```
    }
```

5.3.2 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 );
      }
    }
```

5.3.3 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)
            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.exit(1);
          }
        }
        Jobclient. runjob (job );
      }
    }
```

5.3.4 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;
    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.") ;
        }
      }
    }
```

5.3.5 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: secondarysrot <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 (schemeiutils. 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);
```

5.3.6 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.

- 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.BufferedInputStream;
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 is 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.

5.3.7 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 the 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);
      }
    }
```

5.3.8 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_ou
t hello;
```

Expected output

The content of output table “mr_grep_out” is as follows:

```
+------------+------------+
| key | value |
+------------+------------+
| 2 | hello |
+------------+------------+
```

Sample code

```
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);
      }
    }
```

5.3.9 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.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, the data for the left table
is all in memory.
//This implementation is just a functional display with relatively low
 performance and is not recommended 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 );
      }
    }
```

5.3.10 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
reductor number to 0.
    job.setNumReduceTasks(0);
    // Because there is no input table, the number of mapper needs to
be specified explicitly by the user
    job.setNumMapTasks(1);
    job.set(SLEEP_SECS, args[0]);
    JobClient.runJob(job);
  }
}
```

5.3.11 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 set
        /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.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 );
        }
      }
    }
```

5.3.12 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 schemeiutils.
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.") ;
      }
```

```
    }
```

5.3.13 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.

5.3.14 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 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 ， OutputGrou
pingColumns,
         * 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
) })
            .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
assemblyer
        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);
      }
    }
```

5.4 Java SDK

5.4.1 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
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.

Interface	Description
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
void setResources(String resourceNames)	Declares resources used in this job. Only the declared resource can be read by TaskContext object during Mapper/Reducer running process .
void setMapOutputKeySchema(Column[] schema)	Sets the Key attribute output from Mapper to Reducer.
void setMapOutputValueSchema(Column[] schema)	Sets the Value attribute output from Mapper to Reducer.
void setOutputKeySortColumns(String[] cols)	Sets key sort columns output from Mapper to Reducer.
void setOutputGroupingColumns(String[] cols)	Sets Key grouping columns.
void setMapperClass(Class<? extends Mapper > theClass)	Sets Mapper function of the job.
void setPartitionColumns(String[] cols)	Sets the partition column specified in the job . The default is all columns of Key output by Mapper.
void setReducerClass(Class<? extends Reducer theClass)	Sets Reducer of the job.
void setCombinerClass(Class<? extends Reducer theClass)	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.
void setSplitSize(long size)	Sets the size of input slice. Unit: MB. The default value is 640.
void setNumReduceTasks(int n)	Sets the number of Reducer tasks. The default is 1/4 of Mapper tasks.
void setMemoryForMapTask(int mem)	Sets the memory size of single Worker in the Mapper task. Unit: MB. The default value is 2048.
void setMemoryForReduceTask(int mem)	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
static RunningJob runJob(JobConf job)	Returns immediately after submitting a MapReduce job in a synchronous (blocking) mode.
static RunningJob submitJob(JobConf job)	Returns immediately after submitting a MapReduce job in an asynchronous (non-blocking) mode.

RunningJob

Main function interfaces are as follows.

Interface	Description
String getInstanceID()	Gets an instance ID for checking run log and job management.
boolean isComplete()	Checks whether job is complete.
boolean isSuccessful()	Checks whether job instance is successful.
void waitForCompletion()	Waits until job instance is complete. It is typically iused for jobs submitted is asynchrono us mode.
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

5.4.2 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
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

Type	Interface	Is it compatible?
TaskAttemptContext	setStatus(String msg)	Empty implementation
TaskAttemptContext	getStatus()	Empty implementation
TaskAttemptContext	getTaskAttemptID()	No, throw exception
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

Type	Interface	Is it compatible?
Job	getInstance(Configuration conf, String jobName)	Empty implementation
Job	getInstance(JobStatus status, Configuration conf)	No, throw exception
Job	getJobFile()	No, throw exception
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

Type	Interface	Is it compatible?
Job	getTrackingURL()	Yes
Job	isComplete()	Yes
Job	isRetired()	No, throw exception
Job	isSuccessful()	Yes
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

Type	Interface	Is it compatible?
Job	setJobName(String name)	Empty implementation
Job	setJobSetupCleanupNeeded(boolean needed)	Empty implementation
Job	setMapOutputKeyClass(Class<? > theClass)	Yes
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

Type	Interface	Is it compatible?
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
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.

Type	Interface	Is it compatible?
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
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

Type	Interface	Is it compatible?
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
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

Type	Interface	Is it compatible?
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
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

Type	Interface	Is it compatible?
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
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

Type	Interface	Is it compatible?
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
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

Type	Interface	Is it compatible?
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
JobConf	setMaxPhysicalMemoryForTask(long mem)	Empty implementation
JobConf	setMaxReduceAttempts(int n)	Empty implementation

Type	Interface	Is it compatible?
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
JobConf	setProfileEnabled(boolean newValue)	Empty implementation

Type	Interface	Is it compatible?
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
TextInputFormat	Not involved	Yes
InputSplit	mapred.min.split.size.	No, throw exception

Type	Interface	Is it compatible?
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
OutputCommitter	setupJob(JobContext jobContext)	No, throw exception

Type	Interface	Is it compatible?
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	setDisplayName(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
DistributedCache	addArchiveToClassPath(Path archive, Configuration conf)	No, throw exception

Type	Interface	Is it compatible?
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

Type	Interface	Is it compatible?
Job Authorization	mapred.acls.enabled	No, throw exception
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
Multi{anchor:_GoBack}pleOutputs	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

5.5 MR limits

In order to avoid that you have not paid attention to restrictions so that business stops after the business starts , this article will summarize the MaxCompute MR restrictions to help you.

The restrictions of MaxCompute MapReduce are as follows:

Restricted item	Value	Type	Configuration item	Default value	Configu le?	Description
Memory occupied by the instance	[256MB ,12GB]	Memory limit	odps.stage. mapper(reducer).mem and odps .stage.mapper (reducer).jvm. mem	2048M + 1024M	Yes	Memory occupied by a single map instance or reduce instance, including the framework memory (2 ,048 MB by default) and heap memory of the Java virtual machine (JVM) (1, 024 MB by default).
Number of resources	256	Number limit	N/A	None.	No	The number of resources referenced by a single job cannot exceed 256. The table and archive are regarded as a unit.
Numbers of inputs and outputs	1024 and 256	Number limit	N/A	None	No	The number of inputs of one job cannot exceed 1024. (A partition of a table is regarded as one input. The number of input tables cannot exceed 64). The number of outputs of one job cannot exceed 256.
Number of counters	64	Number limit	N/A	None.	No	The number of custom counters in one job cannot exceed 64. The group name and counter name of a counter must not contain #. The overall length of the group name and the counter name of a counter must be within 100.

Restricted item	Value	Type	Configuration item	Default value	Configu le?	Description
map instance	[1, 100000]	Number limit	odps.stage.mapper.num	None	Yes	The number of map instances of one job is calculated by the framework based on the split size. If no input table exists, you can set the value directly in odps.stage.mapper.num. The final number ranges from 1 to 100,000.
reduce instance	[0,2000]	Number limit	odps.stage.reducer.num	None	Yes	The number of reduce instances of one job is 1/4 of that of map instances by default. The reduce instance number configured by the user ranges from 0 to 2,000. It may occur that the data volume processed by reduce is several times that processed by map. In this case, the reduce phase gets slower and can initiate at most 2000 instances.
Number of retries	3	Number limit	N/A	None	No	The maximum number of retries allowed for a single map instance or reduce instance is 3. Some exceptions that do not allow retries may cause task execution failures.

Restricted item	Value	Type	Configuration item	Default value	Configu le?	Description
Local debug mode	100	Number limit	N/A	None	No	In local debug mode, the number of map instances is 2 by default and cannot exceed 100. The number of reduce instances is 1 by default and cannot exceed 100. The number of download records of one input is 1 by default and cannot exceed 100.
Number of times of reading a resource repeatedly	64	Number limit	N/A	None	No	The number of times that a map instance or reduce instance reads one resource repeatedly cannot exceed 64 .
Resource length	2G	Length limit	N/A	None	No	The total length of a resource referenced by a job cannot exceed 2 GB.
split size	[1,)	Length limit	odps.stage. mapper.split.size	256M	Yes	The framework splits the map based on the configured split size, of which the number of maps is then determined.
Content length of the string column	8 MB	Length limit	N/A	None	No	The content in the string column of the MaxCompute table cannot exceed 8 MB.
Worker running timeout period	[1, 3600]	Time limit	odps.function. timeout	600	Yes	Timeout period for the worker when the map or reduce worker does not read or write data or actively send heartbeat data by using context. progress(). The default value is 600s.

Restricted item	Value	Type	Configuration item	Default value	Configu le?	Description
The supported field types of table referenced by MR	BIGINT、DOUBLE、STRING、DATETIME、BOOLEAN	Data type limit	N/A	None	No	When the MR task refers to a table, an error occurs if the table contains other types of fields.
Read data from OSS		Feature limit	N/A	None	No	Not supported
MaxCompute 2.0 new types		Feature limit	N/A	None	No	Not supported

6 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()
```

7 External table

7.1 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.

7.2 OSS STS mode authorization

This article introduces you how to customize the permissions of MaxCompute to access OSS in RAM.

The location access OSS account supports the incoming plaintext `AccessKeyId` and `AccessKeySecret` when creating the external table, but there is a risk of leaking the account. In some scenarios, this risk is intolerable, so MaxCompute provides a more secure way to access OSS.

MaxCompute combines Alibaba Cloud's Access Control Service (RAM) and Token Service (STS) to address account security issues. You can grant permissions in two ways:

- When the owner of MaxCompute and OSS are the same account, a one-click authorization operation can be performed directly on the RAM console.
- Custom authorization.

  1. The first thing you need to authorize MaxCompute to access the OSS permissions in RAM. Create a role, and the role name such as `AliyunODPSDefaultRole` or `AliyunODPS RoleForOtherUser` , and set the policy content:

```
-- When the owner of MaxCompute and OSS are the same account

"Statement ":[

"Action": "STS: apererole ",
"Effect": "allow ",
"Principal ":{
"Service ":[
"Maid"

}


"Version": "1"

-- When the owner of MaxCompute and OSS are not the same account

"Statement ":[

"Action": "STS: apererole ",
"Effect": "allow ",
"Principal ":{
"Service ":[
"MaxCompute's owner cloud account page"




"Version": "1"
```

  2. Grant the role the necessary permissions to access the OSS * . As follows:

```
Version: "1 ",
"Statement ":[

"Action ":[
 "Oss: listbuckets ",
 "Oss: GetObject ",
 "Oss: maid ",
 "Oss: putobject ",
 "Oss: deleteobject ",
 "Oss: maid ",
 "Oss: listparts"

"Resource ":"*",
```

```
"Effect": "allow"


-- Can Customize other Permissions
```

3. The permission box is then granted to the role.

Note:
After the authorization is complete, view the role details to obtain the Ran information of the Role. You need to specify this Ran information when you create the OSS external table.

7.3 Access OSS unstructured data

This article will show you how to easily access OSS data on MaxCompute.

Authorization with STS Mode

Authorize OSS data permission to MaxCompute account in advance, in order that MaxCompute can directly access the OSS. You can authorize permissions in the following two ways:

- *When the MaxCompute and OSS owner are the same account*, you can directly log on Alibaba Cloud account and *click here to complete authorization*.
- Custom authorization.

1. Firstly, you must authorize MaxCompute permission to access OSS in *RAM*. Log on to the *RAM console* (if MaxCompute and OSS are not identical, authorized by OSS account login) and create roles through *role management* in the console, such as `AliyunODPS DefaultRole` or `AliyunODPSRoleForOtherUser`.
2. Modify the policy content of role as follows:

```
--When the MaxCompute and OSS owner are the same account:
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
--When the MaxCompute and OSS owner are not the same account:
{
"Statement": [
{
 "Action": "sts:AssumeRole",
```

```
 "Effect": "Allow",
 "Principal": {
   "Service ":[
     "MaxCompute's Owner account: id@odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
```

3. The necessary permission `AliyunODPSRolePolicy` granted to the role to access OSS is shown below.

```
{
"Version": "1",
"Statement": [
{
 "Action": [
   "oss:ListBuckets",
   "oss:GetObject",
   "oss:ListObjects",
   "oss:PutObject",
   "oss:DeleteObject",
   "Oss: maid ",
   "oss:ListParts"
 ],
 "Resource": "*",
 "Effect": "Allow"
}
]
}
--You can customize other permissions.
```

4. Authorize the permission **AliyunODPSRolePolicy** to this role.

Read OSS Data with Built-in Extractor

When accessing external data sources, you must use different custom Extractor. You can also use MaxCompute's build-in Extractor to read conventionally-formatted data stored in *OSS*. You only need to create an external table and use this table as the source table for query operations.

In this example, assume that you have a CSV data file in *OSS*. The endpoint is `oss-cn-shanghai-internal.aliyuncs.com`, the bucket is `oss-odps-test`, and the data file is stored in */demo/vehicle.csv*.

Creating an External table

Use the following statements to create an external 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' -- (1)
WITH SERDEPROPERTIES (
 'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
) -- (2)
LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
Demo/'; -- (3)(4)
```

The above statement is described below:

- `com.aliyun.odps.CsvStorageHandler` is the built-in `StorageHandler` for processing CSV-format files. It defines how CSV files are read and written. You only need to specify this name. The relevant logic is implemented by the system.
- The information in `odps.properties.rolearn` comes from the `Arn` information of `AliyunODPSDefaultRole` in RAM. You can get it through the *role details* in the RAM console.
- You must specify an OSS directory for LOCATION. By default, the system reads all the files in this directory.
  - We recommend you to use the domain name of the intranet, to avoid incurring fees for the OSS data-flow.
  - We recommend that the region you store the OSS data is the same as the region you open MaxCompute. Because MaxCompute can only be deployed in some regions, cross-regional data connectivity cannot be guaranteed.
  - OSS connection format is `oss://oss-cn-shanghai-internal.aliyuncs.com/bucketname/directoryname/`. You do not have to add a file name after the directory. Some common errors are shown as follows:

    ```
    http://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo/
     -- HTTP connection is not supported.
    https://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo/
     -- HTTPS connection is not supported.
    oss://oss-odps-test.oss-cn-shanghai-internal.aliyuncs.com/Demo
     -- The connection address is incorrect.
    oss://oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
    Demo/vehicle.csv  -- You do not need to specify the file name.
    ```
- In the MaxCompute system, external tables only record the associated OSS directory. If you DROP (delete) this table, the corresponding `LOCATION` data is not deleted.

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 tables information such as StorageHandler and Location.

Access Table Data by Using an External Table

After creating an external table, you can use it as a normal table. Assume the data in `/demo/vehicle.csv` is:

```
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
```

Run the following SQL statement:

```
select recordId, patientId, direction from ambulance_data_csv_external
 where patientId > 25;
```

Note:

Currently, external table can only be operated through MaxCompute SQL. MaxCompute MapReduce cannot operate the external table.

This statement submits a job, scheduling the built-in CSV extractor to read and process data from OSS. The result is as follows:

```
+------------+------------+-----------+
| recordId   | patientId  | direction |
+------------+------------+-----------+
| 1 | 51 | S |
| 3          | 48         | NE        |
| 4          | 30         | W         |
| 5          | 47         | S         |
| 7          | 53         | N         |
| 8          | 63         | SW        |
| 10         | 31         | N         |
+------------+------------+-----------+
```

Read OSS Data Using a Custom Extractor

When OSS data is in a complex format, and the built-in Extractor cannot meet your requirements, you must use a custom extractor to read data from OSS files.

For example, assume you have a txt data file that is not in CSV format, and `|` is used as the column delimiter between records. For example, the data in `/demo/SampleData/CustomTxt/AmbulanceData/vehicle.csv` is:

```
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
```

- Define an Extractor

  Write a common extractor, using the delimiter as the parameter. This allows you to process all text files with similar formats. As follows:

```
/**
 * 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 is an InputStreamSet, each call
to next() returns an InputStream. This InputStream can read all the
content in an OSS file.
    this.attributes = attributes;
    // check if "delimiter" attribute is supplied via SQL query
    String columnDelimiter = this.attributes.getValueByKey("
delimiter"); //The delimiter parameter is supplied by a DDL
statement.
    if ( columnDelimiter ! = NULL)
    {
      this.columnDelimiter = columnDelimiter;
    }
    // note: more properties can be inited from attributes if needed
  }
  @Override
  public Record extract() throws IOException {//extractor() calls
return one record, corresponding to one record in an external table.
    String line = readNextLine();
    if (line == null) {
```

```
      return null;  // A return value of NULL indicates that this
table has no readable records.
    }
    return textLineToRecord(line);  // textLineToRecord splits a row
 of data into multiple columns according to the delimiter.
  }
  @Override
  public void close(){
    // no-op
  }
}
```

See *here* for a complete implementation of textLineToRecord splitting data.

- Define StorageHandler

  A StorageHandler acts as a centralized portal for custom External Table logic.

```
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 Outputer>getOutputerClass() {
    return TextOutputer.class;
  }
}
```

- Compiling and Packaging

  Compile your custom code into a package and upload it to MaxCompute.

```
add jar odps-udf-example.jar;
```

- Create External Table

  Similar to using the built-in Extractor, first, you must create an external table. The difference is that, when specifying the external table access data, you must use a custom StorageHandler.

  Use the following statements to create an external table:

```
CREATE EXTERNAL TABLE IF NOT EXISTS ambulance_data_txt_external
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.udf.example.text.TextStorageHandler'   --
STORED BY specifies the custom StorageHandler class name.
  with SERDEPROPERTIES (
```

```
'delimiter'='\\|', -- SERDEPROPERITES can specify parameters, these
 parameters are passed through the DataAttributes to the Extractor
code.
'odps.properties.rolearn'='acs:ram::xxxxxxxxxxxx:role/aliyunodps
defaultrole'
)
LOCATION 'oss://oss-cn-shanghai-internal.aliyuncs.com/oss-odps-test/
Demo/SampleData/CustomTxt/AmbulanceData/'
USING 'odps-udf-example.jar'; --You must also specify the jar
package containing the class definition.
```

- Query an External Table

  Run the following SQL statement:

```
select recordId, patientId, direction from ambulance_data_txt_e
xternal where patientId > 25;
```

Read Unstructured Data by Using a Custom Extractor

Previously, you can use the built-in Extractor or a custom Extractor to conveniently process CSV and other text data stored in OSS. Next, using audio data (wav format files) as an example, the following shows how to use a custom Extractor to access and process non-text files in OSS.

Here, starting from the last SQL statement, we introduce the use of MaxCompute SQL as a portal to process audio files stored in OSS.

Create the External table SQL as follows:

```
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';
```

As in the preceding example, you must create an external table. Then, use the schema of this table to define the information that you want to extract from the audio file:

- The statement signal-to-noise ratio(SNR) in an audio file: sentence_snr.
- The name of the audio file: id.

After creating the external table, use a standard Select statement to perform a query. This operation triggers the Extractor to perform computation. When reading and processing OSS data, in addition to simple deserialization on text files, you can use custom Extractor to perform more complex data processing and extraction logic. In this example, use the custom extractor

encapsulated in `com.aliyun.odps.udf.example.speech.SpeechStorageHandler` to calculate the average SNR of valid statements in the audio file, and extract structured data for SQL operations (WHERE sentence_snr > 10). Once completed, the operation returns all audio files with an SNR that greater than 10 and their corresponding SNR values.

Multiple WAV-format files are stored at the OSS address `oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/dev/SpeechSentenceTest/`. The MaxCompute framework reads all the files at this address and, when necessary, performs file-level sharding. It automatically allocates the file to multiple computing nodes for processing. On each computing node, the extractor is responsible for processing the file set allocated to the node by InputStreamSet. The special processing logic is similar to your single-host program. Your algorithm is implemented by using the single host method according to its class.

Details about the `SpeechSentenceSnrExtractor` formulation logic are as follows:

First, read the parameters in the `setup` interface to perform initialization and import the audio processing model (using resource introduction):

```
public SpeechSentenceSnrExtractor(){
    this.utteranceLabels = new HashMap<String, UtteranceLabel>();
  }
  @Override
  public void setup(ExecutionContext ctx, InputStreamSet inputs,
DataAttributes attributes){
    this.inputs = inputs;
    This. Attributes = attributes;
    this.mlfFileName = this.attributes.getValueByKey(MLF_FILE_A
TTRIBUTE_KEY);
    String sampleRateInKHzStr = this.attributes.getValueByKey(
SPEECH_SAMPLE_RATE_KEY);
    this.sampleRateInKHz = Double.parseDouble(sampleRateInKHzStr);
    try {
      // read the speech model file from resource and load the model
into memory
      BufferedInputStream inputStream = ctx.readResourceFileAsStream(
mlfFileName);
      loadMlfLabelsFromResource(inputStream);
      inputStream.close();
    } catch (IOException e) {
      throw new RuntimeException("reading model from mlf failed with
exception " + e.getMessage());
    }
  }
```

The extract() interface implements reading and processing logics of the voice file, computes the signal-to-noise ratio (SNR) of the data based on the voice model, and fills Record with the result in the format of [snr, id].

The preceding example simplifies the implementation process and does not include the relevant audio processing algorithm logic. see the *example code* provided by the MaxCompute SDK in the open source community.

```
@Override
  public Record extract() throws IOException {
    SourceInputStream inputStream = inputs.next();
    if (inputStream == null){
      return null;
    }
    // process one wav file to extract one output record [snr, id]
    String fileName = inputStream.getFileName();
    fileName = fileName.substring(fileName.lastIndexOf('/') + 1);
    logger.info("Processing wav file " + fileName);
    String id = fileName.substring(0, fileName.lastIndexOf('.'));
    // read speech file into memory buffer
    long fileSize = inputStream.getFileSize();
    byte[] buffer = new byte[(int)fileSize];
    int readSize = inputStream.readToEnd(buffer);
    inputStream.close();
    // compute the avg sentence snr
    double snr = computeSnr(id, buffer, readSize);
    // construct output record [snr, id]
    Column[] outputColumns = this.attributes.getRecordColumns();
    ArrayRecord record = new ArrayRecord(outputColumns);
    record.setDouble(0, snr);
    record.setString(1, id);
    return record;
  }
  private void loadMlfLabelsFromResource(BufferedInputStream
fileInputStream)
          throws IOException {
    //  skipped here
  }
  // compute the snr of the speech sentence, assuming the input buffer
 contains the entire content of a wav file
  private double computeSnr(String id, byte[] buffer, int validBuffe
rLen){
    // computing the snr value for the wav file (supplied as byte
buffer array), skipped here
  }
```

Run the query:

```
select sentence_snr, id
    from speech_sentence_snr_external
where sentence_snr > 10.0;
```

Results:

```
-------------------------------------------------------------
| 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   |
-------------------------------------------------------------
```

By using the customized Extractor, you can process multiple voice data files stored on OSS on the SQL statement in a distributed way. The same method can also easily use the large-scale computing power of MaxCompute to complete the processing of various types of unstructured data such as images, videos and so on.

Partition of data

In earlier sections, an external table linked data is implemented through designated OSS Directory on LOCATION. But while process, MaxCompute reads all data under Directory, *including all files in sub-directory*. For accumulated data directories along with time, because the data volume is too big, scan the entire directory may cause unnecessary extra IO and data processing time. Normally, there are 2 solutions for this problem.

- Reducing the volume of access data: You can plan the data storage address, consider using multiple EXTERNAL TABLE to describe different parts of the data, and let each EXTERNAL TABLE LOCATION point to a subset of the data.
- Partition data: EXTERNAL TABLE is the same as internal table, *it supports functions of partition table*, you are available to manage data systemization based on partition function.

It mainly introduces partition function of EXTERNAL TABLE in this section.

- *Standard Organization Method and Path Format of Partition Data on OSS*

  Unlike its internal tables, MaxCompute does not have the authority to manage data stored in external memory (such as OSS). As such, if you must use the partition table function on your system, the storage path for data files on OSS needs to conform to a certain format. This format is as follows.

  ```
  partitionKey1=value1\partitionKey2=value2\...
  ```

  Related examples are as follows

  Assume that you save your daily LOG files on OSS and want to access part of the data when processed with MaxCompute, based on the granularity of Day. Assuming that these LOG files are CSV files (usage of complicated and customized format is similar), you can define data using the following *partitioned external 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/aliyunodpsdefaultrol
e'
)
  LOCATION 'oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/
log_data/';
```

Like the previous table statement, the difference with the previous example is that when you define an external table, the external table is specified as a partition table through the PARTITIONED BY syntax, and the example is a three-tier partition table, the key for the partition is year, month, and day.

In order for a partition like this to work effectively, you must comply with the aforementioned path format when storing data on OSS. The following is an example of a valid path storage layout.

```
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:

If you have prepared the offline data, that is, if you have uploaded the offline data to the OSS storage service with **osscmd** or other OSS tools, you then define the data path format.

You can introduce the partition information into maxcompute by using the **ALTER TABLE ADD PARTITIONDDL** pant statement.

An example of the corresponding DDL statement is as follows.

```
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:

These actions are the same as the standard maxcompute internal table operation, and for more information about the partition, see*Partition*. When the data is ready and the PARTITION information has been imported into the system, the partitioning of the external table data on OSS can be performed by means of an SQL statement.

Assuming that you only want to analyze how many different IPs there are in LOG on June 1, 2016, the following command can be used:

```
SELECT count(distinct(ip)) FROM log_table_external WHERE year = '
2016' AND month = '06' AND day = '01';
```

At this point, for log_table_external, the directory that corresponds to the external table will only access the files under the `log_data/year=2016/month=06/day=01` subdirectory (logfile and logfile 1), *not on the whole* log_data/to avoid a large number of useless I/O operations.

Similarly, if you only want to analyze the data for the second half of 2016, you can use the following command:

```
SELECT count(distinct(ip)) FROM log_table_external
WHERE year = '2016' AND month > '06';
```

At this point, only access the second half of the LOG stored on OSS.

- *Customized Path of Partition Data on OSS*

  If you have historical data stored on OSS but it is not stored using the `partitionKey1 =value1\partitionKey2=value2\...` path format, you can still access it using MaxCompute's partition mode. MaxCompute also provides a way to import partitions through a customized path.

  Assume that only a simple partition value is on your data path (and no partition key information). The following is an example of the data path storage layout.

```
osscmd ls oss://oss-odps-test/log_data_customized/
2017-01-14 08:03:35 128MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/06/01/logfile
2017-01-14 08:04:12 127MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/06/01/logfile. 1
2017-01-14 08:05:02 118MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/06/02/logfile
2017-01-14 08:06:45 123MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/07/10/logfile
2017-01-14 08:07:11 115MB Standard oss://oss-odps-test/log_data_c
ustomized/2016/08/08/logfile
```

```
...
```

The external table builder DDL can see the previous example and also specify the partition key in the clause.

To bind different subdirectories to different partitions, you can do so by using a command similar to the following customized partition path.

```
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/';
```

When LOCATION information is added in ADD PARTITION to customize a partition data path. Even if the data is not stored in the recommended format of `partitionKey1=value1\partitionKey2=value2\...`, you can still access the partition data of the subdirectory.

7.4 Processing open source format data for OSS

This article will show you how to process various popular open source data formats (ORC, PARQUET, SEQUENCEFILE, RCFILE, AVRO and TEXTFILE) stored on OSS through unstructured frameworks in MaxCompute.

Accessing the OSS unstructured data shows you how to access the text stored on the OSS on MaxCompute, audio, image, and other format data. The non-structural framework directly calls the implementation of the open source community to parse the open source data format, and seamlessly with the MaxCompute system.

Note:

Before processing the Open Source format data for OSS, it is necessary to authorize *STS mode for OSS*.

Create External Table

The MaxCompute unstructured data framework is associated with a variety of data through external table, external of Open Source format data associated with OSS

```
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}/';
```

Note:

The syntax format is quite similar to hive's syntax, but the following issues need to be noted:

- STORED AS keyword, which is not STORED BY keyword used for ordinary unstructured appearance in this grammar format, is unique in reading open source compatible data at present.

  STORED AS is followed by file format names, such as *ORC/PARQUET/RCFILE/SEQUENCEFILE/TEXTFILE*.
- The column schemas of the external tables must match the schema where the stored data is stored on the specific OSS.
- `ROW FORMAT SERDE` option is not required, and is only available in a number of special formats, for example, textfile needs to be used.
- When **`WITH SERDEPROPERTIES`** associates OSS privileges with *STS mode authorization*, this parameter is required to specify the **`odps.properties.rolearn`** attribute, whose value is the Role Arn information specifically used in RAM.

  If you do not use STS mode, you do not need to specify this property to pass in the clear text `AccessKeyId` and the `AccessKeySecret` directly at location.
- Location if you associate OSS with clear AK, write as follows:

  ```
  LOCATION 'oss://${accessKeyId}:${accessKeySecret}@${endpoint}/${
  bucket}/${userPath}/'
  ```
- Accessing the OSS External tables is not currently supported with outer-network Endpoint.
- Currently the STORE AS single file size cannot exceed 3G, split is recommended if the file is too large.

Example of PARQUET data associated with OSS

Assume that some parquet files are stored on an OSS path, and that each file is in parquet format, the schema is stored in 16 columns (4 columns bigint, 4 columns double, and 8 columns string) the data for the build table Div statement is as follows:

```
CREATE EXTERNAL TABLE tpch_lineitem_parquet
(
  Rochelle orderkey bigint,
  l_partkey bigint,
  l_suppkey bigint,
  Rochelle linenumber 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,
  _Comment string
)
STORED AS PARQUET
LOCATION 'oss://${accessKeyId}:${accessKeySecret}@oss-cn-hangzhou-zmf.
aliyuncs.com/bucket/parquet_data/';
```

The default parquet data is not compressed, and if you need to compress parquet data on MaxCompute, you need set `set odps.sql.hive.compatible=true;`. The supported compression types are: SNAPPY, GZIP.

Text data associated with OSS

If the data is stored as TEXTFILE file on OSS in JSON format for each row and organized by multiple directories in OSS, then MaxCompute partition table and data association can be used. An example of DDL statement for partition table is shown below.

```
CREATE EXTERNAL TABLE tpch_lineitem_textfile
(
  l_orderkey bigint,
  l_partkey bigint,
  l_suppkey bigint,
  l_linenumber bigint,
  l_quantity double,
  l_extendedprice double,
  l_discount double,
  l_tax double,
  l_returnflag string,
  Maid string,
  l_shipdate string,
  Rochelle Commission 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} @ fig /';
```

If the sub-directory under the OSS table directory is organized as Partition Name, the example is as follows.

```
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/'
```

```
...
```

You can ADD PARTITION using the following pant statement.

```
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170102");
ALTER TABLE tpch_lineitem_textfile ADD PARTITION(ds="20170103");
```

If the OSS partition directory is not organized in this way, or not in the table directory at all, the example is as follows:

```
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, you can use the following pant statement to ADD PARTITION.

```
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/';
...
```

Text data supports serdeproperties (key: default)

```
Fields terminator：'\001'
Escape delimitor：'\\'
Collection items terminator：'\002'
Map keys terminator：'\003'
Lines terminate: '\ N'
Null defination：'\\N'
```

CSV data associated with OSS

The Tasmania statement format 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
    ('Separates atorchare' =, ', 'pigeon techar' = '"', 'escarechar '=
 '\\')
STORED AS TEXTFILE
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

As you can see from the above statement, the CSV data pant Statement supports serdeproperties (key: default)

```
separatorChar：','
```

```
quoteChar：""
Escarechar :'\'
```

Note:

hive OpenCSVSerde only supports string types.

OpenCSVSerde does not currently belong to Builtin Serde. When DML statements are executed, you need to set odps.sql.hive.compatible = true.

JSON data associated with OSS

The Tasmania statement format is as follows, and SERDEPROPERTIES is supported.

```
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}/';
```

ORC data associated with OSS

The Tasmania statement format is as follows.

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED AS ORC
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

AVRO data associated with OSS

The format of the DDL statement is as follows.

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED AS AVRO
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

SEQUENCEFILE data associated with OSS

```
CREATE EXTERNAL TABLE [IF NOT EXISTS]
(<column schemas>)
[PARTITIONED BY (partition column schemas)]
STORED AS SEQUENCEFILE
```

```
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/';
```

Read and process open source format data for OSS

Compare the two external representations created in the previous article, you can see that for different file types, Simply modify the format name after STORED AS. In the following example , only the processing of the appearance (tpch_lineitem_parquet) corresponding to the above PARQUET data will be described centrally. If you want to work with different file types, just specify parquet/ORC/TEXTFILE/RCFILE/TEXTFILE as long as you want to create the appearance when the DDL is created, the statement that processes the data is the same.

- *Read and process open source data directly from OSS*

  After creating a data table to associate, you can directly do the same thing as a normal MaxCompute table, as shown below.

  ```
  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;
  ```

  The appearance tpch_lineitem_parquet is used as a common internal table, except that the MaxCompute internal computing engine reads the corresponding PARQUET data directly from OSS for processing.

  The external partition table for the associated textfile that was created in the previous article, because`ROW FORMAT`+ `STORED AS` is used, you need to set flag manually (Only use STORED AS, odps.sql.hive.compatible is FALSE by default.) and then reads again, otherwise there will be an error.

  ```
  SELECT * FROM tpch_lineitem_textfile LIMIT 1;
  Failed: Maid: User Defined Function exception-traceback:
  com.aliyun.odps.udf.UDFException: java.lang.ClassNotFoundException:
  com.aliyun.odps.hive.wrapper.HiveStorageHandlerWrapper
  --You need to manually set up hive compatible flag.
  set odps.sql.hive.compatible=true;
  Select * from Maid limit 1;
  +------------+------------+------------+--------------+------------
  +-----------------+------------+------------+--------------
  +--------------+------------+--------------+--------------
  +----------------+------------+-----------+
  | l_orderkey | l_partkey  | l_suppkey  | l_linenumber | 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:

Direct use of the external table, each time reading data requires I/O operations involving external OSS, and the MaxCompute system itself does not use many high-performance optimizations for internal storage, so there will be a loss in performance. Therefore, if it is a scenario that requires repeated computation of data and is sensitive to the efficiency of computation, it is recommended to use the following usage: first import the data into MaxCompute and then calculate it.

These complex data types are involved in SQL (create, select, insert, etc.). The statement `set odps. sql. type. system. odps2 = true;` should be added before the SQL statement, and the set statement and the SQL statement should be submitted together for execution at execution time. See for details *Data types*.

- *Importing the open source data from OSS into MaxCompute for Calculation*

  First, create an internal table tpch_lineitem_internal, which is the same as the external table schema, and then import the open source data from OSS into the internal table of MaxCompute for storage in the internal data storage format of MaxCompute.

  ```
  CREATE TABLE tpch_lineitem_internal LIKE tpch_lineitem_parquet;
  INSERT OVERWRITE TABLE tpch_lineitem_internal;
  SELECT * FROM tpch_lineitem_parquet;
  ```

  Next take the same action directly to the internal 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_internal
  WHERE l_shipdate <= '1998-09-02'
  GROUP BY l_returnflag, l_linestatus;
  ```

  By doing so, you can pilot the data into the MaxCompute system for storage, computational processing of the same data will be more efficient.

7.5 Unstructured data exported to OSS

The unstructured framework of MaxCompute supports output of MaxCompute data directly to OSS via insert, MaxCompute also associates OSS with external tables for data output.

Accessing OSS unstructured data shows you how MaxCompute can be accessed and processed through the associations of External tables unstructured data stored in OSS.

Output Data to OSS is typically two cases:

- The MaxCompute internal table is output to the External table that is associated with the OSS.
- After MaxCompute processes the external tables, the results are output directly to the external tables that are associated with the OSS.

Like accessing OSS data, MaxCompute supports output via built-in storagehandler and custom storagehandler.

Output to OSS via built-in StorageHandler

Using the built-in StorageHandler in MaxCompute And can be very convenient to output data in the agreed format to the OSS for storage. All we need to do is create an external table that indicates the built-in StorageHandler, it can be associated with this table, and the related logic is implemented by the system.

Currently MaxCompute supports 2 built-in StorageHandler:

- com.aliyun.odps.CsvStorageHandler , Defines how to read and write CSV format data, data format Conventions: English comma, column separator`,` line Break is `\n`.
- com.aliyun.odps.TsvStorageHandler, defines how to read and write CSV format data, data format Conventions: `\t`is a column separator, line Break is `\n`.
- *Creating 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}/';
```

  - STORED By, if the data file that is required to be exported to OSS is a TSV file, then built-in `com.aliyun.odps.TsvStorageHandler` ; if the data file that is required to be exported to OSS is a CSV file, a built-in `com.aliyun.odps.CsvStorageHandler`.
  - WITH SERDEPROPERTIES, when associating OSS privileges with "Custom Authorization" of "STS Mode Authorization", this parameter needs to specify

the'odps.properties.rolearn'attribute, whose value is the information of the Custom Role specifically used in RAM.

Note:

STS mode authorization can be seen in *Accessing the unstructured data of OSS*.

- Location that specifies the path to the file that corresponds to the OSS storage. If the 'odps.properties.rolearn' attribute is not set in *WITH SERDEPROPERTIES* and the plaintext AK is used for authorization, then *LOCATION* is

```
LOCATION
    'oss://${accessKeyId}:${accessKeySecret}@${endpoint}/${bucket
}/${userPath}/'
```

- The data is output to OSS through the INSERT operation of External Table.

Note:

The insert-to-OSS single file size cannot exceed 5g.

When associated with an OSS storage path via external table,you can do a standard SQL INSERT OVERWRITE/INSERT INTO operation on the External table to output both data to OSS.

```
INSERT OVERWRITE|INTO TABLE <external_tablename> [PARTITION (
partcol1=val1, partcol2=val2 ...)]
select_statement
FROM <from_tablename>
[WHERE where_condition];
```

- *from_tablename*: it can be an internal table, it can also be an external table (including an external table for the associated OSS or OTS).
- INSERT specifies the format of 'StorageHandler'(that is, TSV or CSV) according to the External table 'stored ') write to OSS.

When the INSERT operation is completed successfully, you can see that the corresponding LOCATION on the OSS produces a series of files.

For example: the location corresponding to External table is the *oss://oss-cn-hangzhou-zmf.aliyuncs.com/oss-odps-test/tsv_output_folder/* , you can see the generation of a series of files in the OSS corresponding path:

```
osscmd 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/20170113224724561g9m6csz7/M1_0_0-0.tsv
```

```
2017-01-14 06:48:05 4.78MB Standard oss://oss-odps-test/tsv_output
_folder/.odps/20170113224724561g9m6csz7/M1_1_0-0.tsv
2017-01-14 06:47:48 4.79MB Standard oss://oss-odps-test/tsv_output
_folder/.odps/20170113224724561g9m6csz7/M1_2_0-0.tsv
...
```

The folder tsv_output_folder under the OSS bucket oss-odps-test specified by LOCATION contains the .odps folder which includes some .tsv files and a .meta file. Similar file structures are specific to MaxCompute's output to OSS:

- When you use MaxCompute to execute INSERT INTO/OVERWRITE on an external table and write to an OSS address, all of the data is writen 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 the valid data in the current folder. Typically, if the INSERT operation is successful , all the data in the current folder is valid. The macro data only needs to be parsed when a job fails. If an operation fails midway or is killed, you can simply re-execute the INSERT OVERWRITE statement.
- If it is a partition table, A corresponding partition sub-directory is generated based on the partition value specified by the insert statement under the fig folder and then the partition sub-directory inside is '.odps 'folder. For example, `test/tsv_output_folder/first-level partition name = partition value/.odps/20170113224724561g9m6csz7/M1_2_0-0.tsv`.

For the TSV/CSV storagehandler processing built in by MaxCompute, the number of files generated is corresponding to the corresponding SQL

If the `INSER OVERWITE ... SELECT ... FROM ... ;`operation allocates 1000 mappers on the source data table (from_tablename), 1000 TSV/CSV files will be generated.

Output to OSS via custom storagehandler

In addition to using the built-in StorageHandler to implement the output TSV/CSV common text format on the OSS, the MaxCompute unstructured framework provides a general-purpose SDK that supports external output of custom data format files.

As well as the built-in StorageHandler, you need to "Create an External TABLE" And then output data to OSS through INSERT operation of External Table. The difference is that when creating external tables, STORED BY is required to specify custom StorageHandler.

Note:

The MaxCompute unstructured framework describes the processing of varieties of data storage formats through an interface called StorageHandler. Specifically, the StorageHandler acts as a Wrapper class, allowing you to specify a custom Exatractor(for data reading, parsing, processing, etc) And Outputer(for data processing and output, etc). Custom StorageHandler should inherit Odps StorageHandler and implement getExtractorClass and getOutputerClass interfaces.

Next, we use the example of *Access OSS data*. OSS unstructured data of "Custom Extractor accesses OSS", to show how MaxCompute can output data to OSS txt file by customizing StorageHandler, and use'|'as column delimiter and'\ n' as line breaker.

Note:

After the MaxCompute Studio is configured with *MaxCompute Java module*, you can see the corresponding sample code in examples. Or click *here* to see the complete code.

- *Define Outputer*

  Both output logic must implement the outputer interface:

```
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.Outputer;
import java.io.IOException;
public class TextOutputer extends Outputer {
    private SinkOutputStream outputStream;
    private DataAttributes attributes;
    private String delimiter;
    public TextOutputer (){
        // 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;
    }
    @Override
    public void close() {
        // no-op
    }
    private String recordToString(Record record){
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i &lt; 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();
    }
}
```

All output logic must call the Outputer API. There are three outputer APIs (setup, output, and close) which all correspond to the three extractor APIs (setup, extract, and close). Where setup () and close() are called only once in an outputer. The user may perform initialization preparatio n in setup. Furthermore, the three parameters returned by setup() must be saved as outputer class variables to be used in the output() or close() APIs. The interface, close (), is used to sweep the end of the Code.

Typically, most of the data processing occurs in the output (Record) interface. The MaxCompute system calls output (Record) Once based on each input record processed by the current outputer assignment). Assuming that when an output (Record) call returns, the Code has already consumed the Record, so after the current output (Record) returns, the system uses the memory used by the record for it, so when the information in Record is used across multiple output() function calls, the record for the current process needs to be invoked.clone() method to save the current record.

- *Define Extractor*

  Exatrractor is used for Data Reading, parsing, processing, and so on, if the output tables eventually do not need to be read by MaxCompute and so on, you do not need to define them.

```
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 inited 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 IllegalArg
umentException
    {
        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 &lt; parts.length; i++){
                // only parse data in columns indexed by output
indexes
```

```
                if (index &lt; 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 *Accessing the OSS unstructured data* documentation.

- *Define 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.Outputer;
public class TextStorageHandler extends OdpsStorageHandler {
    @Override
    public Class&lt;? extends Extractor&gt; getExtractorClass() {
        return TextExtractor.class;
    }
    @Override
    public Class&lt;? extends Outputer> getOutputerClass() {
        return TextOutputer.class;
    }
}
```

If the table does not need to be read, you do not need to specify an Extractor interface.

- *Compile and package*

Compile your custom code into a package and upload it to MaxCompute. If the jar package is named 'odps-TextStorageHandler.jar', upload to MaxCompute

```
add jar odps-TextStorageHandler.jar;
```

- *Creating External tables*

Like using the built-in StorageHandler, an External table needs to be created, the difference is that this time you need to specify that the data is output to an external table, using a custom StorageHandler.

```
CREATE EXTERNAL TABLE IF NOT EXISTS output_data_txt_external
(
vehicleId int,
recordId int,
patientId int,
calls int,
locationLatitute double,
locationLongtitue double,
recordTime string,
direction string
)
STORED BY 'com.aliyun.odps.examples.unstructured.text.TextStorag
eHandler'
WITH SERDEPROPERTIES(
    'delimiter'='|'
    [,'odps.properties.rolearn'='${roleran}'])
LOCATION 'oss://${endpoint}/${bucket}/${userfilePath}/'
```

```
USING 'odps-TextStorageHandler.jar';
```

Note:

If you need `odps.properties.rolearn` property, for more information, see *custom authorization* for STS mode authorization of *Access the OSS unstructured data*. If not, you can refer to *one-click authorization* or use clear-text AK on top of location.

- *Write unstructured files into External Table using INSERT*

  After creating an external table Association on the OSS storage path by customizing the storagehandler, you can do a standard SQL insert override/insert into operation on the External table to output both data to OSS, in the same way as the built-in storagehandler:

```
INSERT OVERWRITE|INTO TABLE &lt;external_tablename&gt; [PARTITION (
partcol1=val1, partcol2=val2 ...)]
Select_statement
FROM &lt;from_tablename&gt;
[WHERE where_condition];
```

  When the insert operation is successful,as the built-in StorageHandler, you can see a series of files generated in the OSS corresponding LOCATION path *.odps* folder.

7.6 Access Table Store data

This document introduces how to import data from Table Store to the MaxCompute computing environment. This allows seamless connections between multiple data sources.

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 TableStore 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`.

The previous article showed you how to *Access OSS unstructured data*.

Both TableStore and MaxCompute have their own type systems. Both Table Store and MaxCompute have their own data type systems. When you process Table Store data in MaxCompute, the data type associations are as follow:

MaxCompute Type	TableStore 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. On this issue, 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

  1. 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.

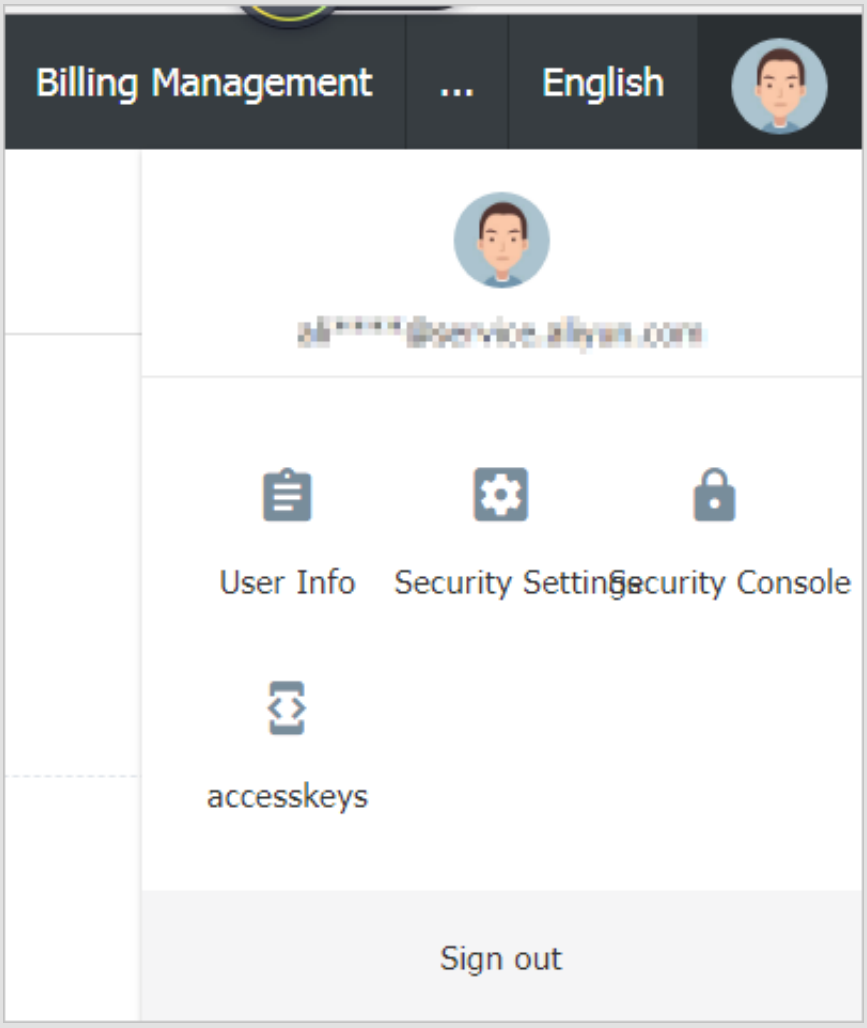
  2. Set its policy content as follows:

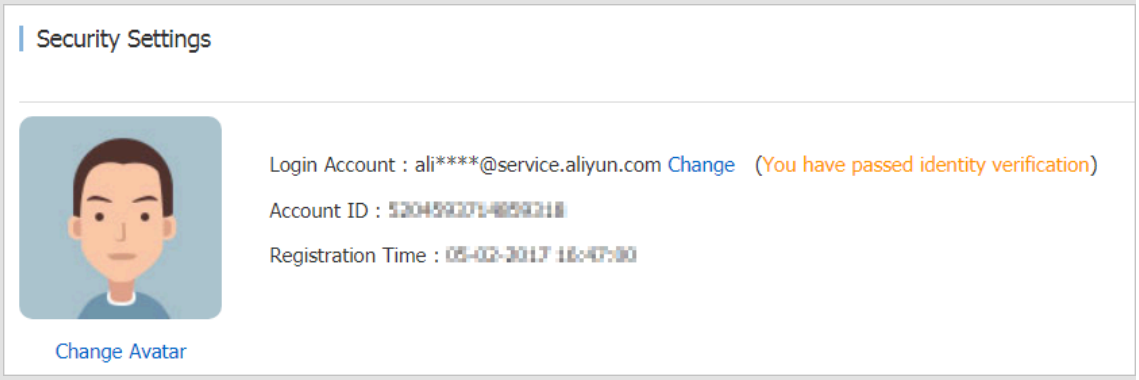
```
--if MaxCompute and Table Store are same account
{
"Statement": [
{
 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "odps.aliyuncs.com"
   ]
 }
}
],
"Version": "1"
}
--if MaxCompute and Table Store are not the same account
{
"Statement ":[
{
 "Action": "sts:AssumeRole",
 "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.





3. 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
```

4. Grant the permission AliyunODPSRolePolicy to this role.

Creating 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', -- ①
'tablestore.table.name'='ots_tpch_orders' -- ②
'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'
 --③
)
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.
- **SERDEPROPERITES** is an interface that provides parameter options. When using **`TableStoreStorageHandler`**, two options must be specified, *`tablestore.columns.mapping`* and *`tablestore.table.name`* and *`odps.properties.rolearn`*.
  1. *`tablestore.columns.mapping option`*: required to describe 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.
  2. *`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.
  3. *`odps.properties.rolearn`*: Arn information in RAM's AliyunODPSDefaultRole. You can get it through the **details of the role** in the RAM console.
- LOCATION clause: specific information for specifying Table Storeinstance names, endpoint, and so on. The secure access to Table Store data here is based on the premise of RAM/STS authorization introduced earlier.

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 tables 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 ecosystem.There, 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 performed in the DDL statement used to create the external table. Certainly, you can retain the original Table Store primary key/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 TableStore

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 already been created (or 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. Now you can write the processed data from internal_orders back to Table Store, perform the `INSERT OVERWITE TABLE` operation on the external table as follows:

```
INSERT OVERWRITE TABLE ots_table_external
SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custk
ey, 'SHIPPED'), CEIL(odps_totalprice)
FROM internal_orders;
```

Note:

If the data in the ODPS table itself has a certain order, such as sorting once according to Primary Key, then when writing to the OTS table, the pressure will be concentrated on an OTS partition, which can not make full use of the characteristics of distributed writing. Therefore, when this happens, we recommend that we first scatter the data through **`distribute by Rand ()`**.

```
INSERT OVERWRITE TABLE ots_table_external
SELECT odps_orderkey, odps_orderdate, odps_custkey, CONCAT(odps_custk
ey, 'SHIPPED'), CEIL(odps_totalprice)
FROM (SELECT * FROM internal_orders DISTRIBUTE BY rand()) t;
```

Because Table Store is a KV data NoSQL storage medium, 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 + odps_orderdate primary key values. In addition , in the 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:

- The data in MaxCompute cannot be written to OTS more than 4 MB at a time, otherwise, the user is required to remove the oversized data and write it back. An error may be generated at this time:

  ```
  ODPS-0010000:System internal error - Output to TableStore failed
  with exception:
  TableStore BatchWrite request id XXXXX failed with error code
  OTSParameterInvalid and message:The total data size of BatchWrite
  Row request exceeds the limit
  ```

- It is a single operation to write data in bulk or by branch. Please refer to *BatchWriteRow* for a detailed description. Therefore, if the volume of bulk write data is too large, you can also branch write.

- When writing data in bulk, be aware that you do not have duplicate rows, otherwise it may cause errors to be reported:

  ```
  Errorcode: FIG, errormessage: the input parameter is invalid
  ```

  For a detailed description, please refer to *using BatchWriteRow to report an OTSParameterInvalid error when submitting 100 pieces of data at a time*.

8 Security

8.1 Target users

This article is intended for MaxCompute project owners, administrators, and users interested in the MaxCompute multi-tenant data security system.

The MaxCompute multi-tenant data security system includes:

- User authentication.
- User and authorization management of projects.
- Sharing of resources across projects.
- Data protection of projects.

8.2 Quick Start

8.2.1 Use case: Add users and grant permissions

Description:

Jack is the project administrator of a project prj1. A new team member named Alice, who already has an Alibaba Cloud account as alice@aliyun.com, applies to join the prj1project. Alice requests the following permissions: view table lists, submit jobs, and create tables.

Solution:

As a project administrator, Jack performs the following procedure to add Alice as the user and grant her permissions to view table lists, submit jobs, and create tables:

```
    use prj1;
    add user aliyun$alice@aliyun.com; --Add the user
    grant List, CreateTable, CreateInstance on project prj1 to user
 aliyun$alice@aliyun.com; --Authorize the user by using the GRANT
statement
```

8.2.2 Use case: Add users and grant permissions using ACL

This article shows you how to add a project role and authorize it through ACLs.

Description:

Jack is the project administrator of a project prj1. The three new data auditors, Alice, Bob, and Charlie, are added to the project team. They all need to apply for the following permissions: view table lists, submit jobs, and read the table userprofile.

Solution:

As a project administrator, Jack can perform authorization by using the object-based *ACL Authorization*.

Jack must perform the following procedure:

```
    use prj1;
    add user aliyun$alice@aliyun.com; --Add the user
    add user aliyun$bob@aliyun.com;
    add user aliyun$charlie@aliyun.com;
    create role tableviewer; --Create a role
    grant List, CreateInstance on project prj1 to role tableviewer; --
Grant permissions to the role
    grant Describe, Select on table userprofile to role tableviewer;
    grant tableviewer to aliyun$alice@aliyun.com; --Grant the
tableviewer role to the user
    grant tableviewer to aliyun$bob@aliyun.com;
    grant tableviewer to aliyun$charlie@aliyun.com;
```

8.2.3 Use case: Project data protection

Description:

Jack is the project administrator of a project prj1. The project involves a large volume of sensitive data including user IDs, shopping records along with the data mining algorithms with proprietary intellectual property rights. Jack wants to protect the sensitive data and algorithms and allow only project users to access the data within the project. He also wants to make sure that data flows within the project only.

Solution:

To protect the project data, Jack must perform these steps:

```
    use prj1;
    set ProjectProtection=true; --Enable the project data protection
mechanism
```

Once the project data protection is enabled, data within the project cannot be transferred out of the project. All the data flows only within the project.

If users want to export data tables out of the project, an approval of the project administrator is needed. Here, MaxCompute provides the TrustedProject configuration to support external data

export from the protected project. In this case, configure project prj2 as a trusted project of prj1 and enable data flow from prj1 to prj2 through the following command:

```
use prj1;
add trustedproject prj2;
```

8.3 User authentication

MaxCompute supports the **Alibaba Cloud account system** and the **RAM account system**.

Note:

MaxCompute recognizes the RAM account system but cannot recognize the RAM permission system. As a user, you can add any of your RAM sub-accounts to a MaxCompute project. However, MaxCompute skips the RAM permission definitions when it verifies the permissions of the RAM sub-account.

By default, the MaxCompute project only recognizes the Alibaba Cloud account system. You can view the account system supported by this project by running `list accountproviders;`.

Typically, only Alibaba Cloud accounts are displayed. To add the RAM account system, run the `add accountprovider ram;` command. After the RAM account system is added, run `list accountproviders;` to make sure it has been successfully added to the supported account systems.

Apply for an Alibaba Cloud account

If you do not have an *Alibaba Cloud account*, visit here to apply for one.

Note:

A valid email address is needed, when you apply for an Alibaba Cloud account. Because this email address is used as the account name after registration. For example, Alice can use her email address alice@aliyun.com to register an Alibaba Cloud account. Her account name will be alice@aliyun.com after Alibaba Cloud account registration.

Apply for AccessKey

Click here to create or manage your *AccessKey* list after you register an Alibaba Cloud account.

An AccessKey consists of the AccessKeyID and AccessKeySecret. The AccessKeyID is used to retrieve the AccessKey, and the AccessKeySecret is used to sign the computing messages. You

must secure your AccessKey for further use. If you need to update an AccessKey, create a new AccessKey and disable the existing one.

Log on to MaxCompute with an Alibaba Cloud account

Configure the AccessKey in the configuration file *conf/odps_config.ini* before you use odpscmd to log on. See the following example:

```
project_name=myproject
 access_id=<Input the AccessKeyID here, excluding the angle brackets>
access_key=<Input the AccessKey here, excluding the angle brackets>
end_point=http://service.odps.aliyun-inc.com/api
```

Note:
To enable or disable an AccessKey on the Alibaba Cloud website, wait for at least 15 minutes after the operation is complete.

8.4 User management

Any user, except the project owner, must be added to the MaxCompute project and granted the corresponding permissions to manage data, jobs, resources, and functions in MaxCompute. This article describes how a project owner can add, authorize, and remove other users, including RAM sub-accounts to MaxCompute.

If you are a project owner, we recommend that you read this article carefully. If you are a typical user, we recommend that you submit an application to the project owner to be added to the corresponding project. We recommend all users to read the subsequent sections.

All the operations mentioned in this article are executed on the console. For Linux, run **./bin/odpscmd** and for Windows, run **./bin/odpscmd.bat**.

Add a user

In this example, the project owner, Alice, wants to authorize another user, therefore she must add the user to the project first. **Only a user who has been added to the project can be authorized**.

The command to add a user is as follows:

```
add user
```

The <username> of an Alibaba Cloud account is a valid email address registered with Alibaba Cloud, or a RAM sub-account of an Alibaba Cloud account that runs the command. For example:

```
add user ALIYUN$odps_test_user@aliyun.com;
```

```
add user RAM$ram_test_user;
```

Assume that the Alibaba Cloud account of Alice is alice@aliyun.com. When Alice runs these statements, the following results are returned by running the `list users;` command:

```
RAM$alice@aliyun.com:ram_test_user
ALIYUN$odps_test_user@aliyun.com
```

This indicates that the Alibaba Cloud account odps_test_user@aliyun.com and the sub-account ram_test_user created by Alice using RAM have been added to the project.

Add a RAM sub-account

The two ways to add a RAM sub-account are as follows:

- By using DataWorks, for more information, see *Prepare a RAM account*.
- By using MaxCompute client commands as described in this document.

> **Note:**
>
> - MaxCompute only allows a primary account to add its own RAM sub-accounts to a project. RAM sub-accounts of other Alibaba Cloud accounts are not allowed. Therefore, you can skip to specify the name of the primary account before the RAM sub-accounts when `add user`. MaxCompute determines by default that the account which runs the command is the corresponding sub-account.
> - MaxCompute only recognizes the RAM account system and does not recognize the RAM permission system. Users can add any of their RAM sub-accounts to a MaxCompute project, but MaxCompute does not consider the permission limits in RAM when performing permission verification of RAM sub-accounts.

By default, MaxCompute project only recognizes Alibaba Cloud account systems. To view the supported account systems use the `list accountproviders;` command. Typically, only the ALIYUN account is visible, for example:

```
odps@ ****>list accountproviders;
ALIYUN
```

> **Note:**
>
> Only the project owner has the permission to perform operations related to `accountproviders`.

As shown in the preceding command, you can only see the `ALIYUN` account system. If you want to add RAM accounts support, run the `add accountprovider ram;` as follows:

```
odps@ odps_pd_inter>add accountprovider ram;
OK
```

The user will still not be able to operate MaxCompute successfully. This is because, the user must be granted certain permissions to operate MaxCompute within the permissive limits. For more information, see *Authorization*.

User Authorization

Once the user is added, the project owner or project administrator must authorize the user. The user can perform the operations only after obtaining the permissions.

MaxCompute provides ACL authorization, cross-project resource sharing, and project resource protection. The following are two common scenarios, for more information, see *ACL Authorization* .

Scenario 1

In the following scenario, Jack is the administrator of the project prj1. A new project team member Alice (Alibaba Cloud account: alice@aliyun.com) applies to join the project prj1, and for permission to view table lists, submit jobs, and create tables.

The admin or the project owner can run the following command on the client:

```
use prj1; --Open the project prj1
add user aliyun$alice@aliyun.com; --Add the user
grant List, CreateTable, CreateInstance on project prj1 to user aliyun
$alice@aliyun.com; --Authorize the user
```

Scenario 2

In the following scenario, assume Alibaba Cloud account user (bob@aliyun.com) has been added to a project ($user_project_name), and must be granted permission to create tables, obtain table information, and run functions.

The admin or the project owner can run the following command on the client:

```
grant CreateTable on PROJECT $user_project_name to USER ALIYUN$bob@
aliyun.com;
  --Grant CreateTable permission on project “$user_project_name” to
bob@aliyun.com
grant Describe on Table $user_table_name to USER ALIYUN$bob@aliyun.com
;
  --Grant Describe permission on table “$user_table_name” to bob@
aliyun.com
grant Execute on Function $user_function_name to USER ALIYUN$bob@
aliyun.com;
```

```
  --Grant Run permission on function "$user_function_name" to bob@
aliyun.com
```

Authorize RAM Sub-account

To check accounts support, run `list accountproviders;` command as follows:

```
odps@ ****>list accountproviders;
ALIYUN, RAM
```

In this project, RAM accounts are also supported. You can add a RAM sub-account to this project and grant `Describe` permission on the tables. For example:

```
odps@ ****>add user ram$bob@aliyun.com:Alice;
OK: DisplayName=RAM$bob@aliyun.com:Alice
odps@ ****>grant Describe on table src to user ram$bob@aliyun.com:
Alice;
OK
```

After running these commands, *`Alice`* account, which is a RAM sub-account of *`bob@aliyun.com`* , can logon to MaxCompute with the **AccessKeyID** and **AccessKeySecret**, and run `desc` on the table *`src`*.

Note:

- For more information about how to create a RAM sub-account `AccessKeyID` and `AccessKeySecret`, see *RCreate a RAM user*.
- For more information about how to add or remove users on MaxCompute, see the correspond ing content of this article.
- For more information about authorizing a user, see *Authorization*.

Remove a User

When a user leaves the project team, Alice must remove the user from the project. Once removed from the project, the user no longer has any access permission to the project resources.

The command to remove a user from a project is as follows:

```
remove user
```

Note:

- A user removed from a project immediately loses an authority to access resources of the project.
- Revoke all the roles of the user, before removing a user whom the roles are assigned. For more information about roles, see *Role Management*.

- After a user is removed, all*ACL Authorization* data related to the user is retained. After a user is added to a project again, the ACL Authorization of this user is enabled again.
- MaxCompute does not support complete removal of a user and all permission data from a project.

To remove corresponding users, Alice can run the following commands:

```
remove user ALIYUN$odps_test_user@aliyun.com;
remove user RAM$ram_test_user;
```

To make sure the users are removed, run the following command:

```
LIST USERS;
```

If those two accounts are no longer listed after running the command, it indicates that the accounts have been removed from the project.

Remove a RAM Sub-account

Similarly, RAM sub-account can be removed by using the `remove user` command. For example:

```
odps@ ****>revoke describe on table src from user ram$bob@aliyun.com:
Alice;
OK
-- Revoke Alice sub-account permissions
odps@ ****>remove user ram$bob@aliyun.com:Alice;
Confirm to "remove user ram$bob@aliyun.com:Alice;" (yes/no)? yes
OK
-- Remove sub-account
```

If you are the project owner, you can also remove the RAM account system from the current project by `remove accountprovider` as follows:

```
odps@ ****>remove accountprovider ram;
Confirm to "remove accountprovider ram;" (yes/no)? yes
OK
odps@ ****>list accountproviders;
ALIYUN
```

8.5 Role management

A role is a defined set of access permissions. It assigns the same set of permissions to a group of users. Role-based authorization greatly simplifies the authorization process and reduces the authorization management cost. It must be used with priority.

When a project is created, an admin role is automatically created with a definite privilege authorized to the role, including access to all objects within the project, management of users

and roles, and authorization to users and roles. In comparison to a project owner, the admin role cannot assign admin permission to any user, set the project security configuration, or change the authentication model for the project. Permissions of the admin role cannot be modified.

Role management related commands include the following:

```
create role <rolename> --Create a role
drop role <rolename> --Delete a role
grant <rolename> to <username> --Grant a role to a user
revoke <rolename> from <username> --Revoke a role from a user
```

Note:

- One role can be assigned to multiple users at the same time, and one user can be assigned multiple roles.
- For more information about the mapping between the roles in DataWorks and in MaxCompute, and the platform permissions of these roles, see the project member management module in *Project Management*.

Create a role

To create a role, use the following command :

```
CREATE ROLE;
```

Example:

To create a role player, enter the following command on the client:

```
create role player;
```

Note:

The role permissions you create can view the specified user permissions through *Permission check*.

Add a user to the role

To add a user to the role, use the following command:

```
GRANT <roleName> TO <full_username> ;
```

Example:

To assign user bob@aliyun.com the player role, enter the following command on the console:

```
grant player to bob@aliyun.com;
```

Authorize role

The authorization statement for the role is similar to the authorization for the user. For more information, see *User authorization*.

Note:
After role authorization is complete, all users under this role have the same permissions.

Example:

Jack is the administrator of project prj1. Three new data auditors, Alice, Bob, and Charlie, are added to the project team. They must apply for the following permissions: view the table lists, submit the jobs, and read the table userprofile.

In this scenario, the project administrator can perform authorization by using the object-based *ACL Authorization*.

The commands are as follows:

```
    use prj1;
    add user aliyun$alice@aliyun.com; --Add the user
    add user aliyun$alice@aliyun.com; --Add the user
    add user aliyun$charlie@aliyun.com;
    create role tableviewer; --Create a role
    grant List, CreateInstance on project prj1 to role tableviewer; --
Grant permissions to the role
    grant Describe, Select on table userprofile to role tableviewer;
    grant tableviewer to aliyun$alice@aliyun.com; --Grant the
tableviewer role to the user
    grant tableviewer to aliyun$bob@aliyun.com;
    grant tableviewer to aliyun$charlie@aliyun.com;
```

Revoke the role from the user

To revoke the role from the user, use the following command:

```
REVOKE <roleName> FROM <full_username>;
```

Example:

To remove the user bob@aliyun.com from the player role, use the following command on the client:

```
revoke player from bob@aliyun.com;
```

Delete a Role

To delete a role, use the following command:

```
DROP ROLE <roleName>;
```

Example:

To delete the role of the player, use the following command:

```
drop role player;
```

Note:
When a role is deleted a role, MaxCompute checks whether other users are in this role. If yes, this role cannot be deleted. The role can be successfully deleted only when all users in the role are revoked from this role.

8.6 Authorization

Authorization allows a user to perform operations including read, write, and view on tables, tasks, resources, and other objects of the MaxCompute. After the *user* is added, the project owner or the project administrator must authorize the user. The user can perform operations only after obtaining the permission.

MaxCompute provides Access Control List (ACL) authorization, cross-project resource sharing, and project resource protection. Authorization typically includes three elements: subject, object, and action. In MaxCompute, the subject refers to a user or a role and the object refers to various types of objects in a project.

ACL authorization includes following MaxCompute objects: *Project*,*Table* ,*Function* , *Resource*, and *Instance*. Operations are related to specific object types, therefore different types of objects support different types of actions.

MaxCompute projects support the following object types and actions:

Object	Action	Description
Project	Read	View project information (excluding any project objects), such as the creation time.

Object	Action	Description
Project	Write	Update project information (excluding any project objects), such as comments.
Project	List	View the list of all types of objects in the project.
Project	CreateTable	Create a table in the project.
Project	CreateInstance	Create an instance in the project.
Project	CreateFunction	Create a function in the project.
Project	CreateResource	Create a resource in the project.
Project	All	Grant all of the preceding permissions.
Table	Describe	Read the metadata of the table.
Table	Select	Read the table data.
Table	Alter	Change the metadata of the table and add or delete a partition.
Table	Update	Overwrite or add table data.
Table	Drop	Delete a table.
Table	All	Grant all the preceding permissions.
Function	Read	Read and run permissions.
Function	Write	Update.
Function	Delete	Delete.
Function	Run	Run.
Function	All	Grant all the preceding permissions.
Resource	Read	Read.
Resource	Write	Update.
Resource	Delete	Delete.
Resource	All	Grant all the preceding permissions.
Instance	Read	Read.
Instance	Write	Update.
Instance	All	Grant all the preceding permissions.

Note:

- The CreateTable action for the objects of Project type must work with the CreateInstance permission for the Project object. The Select, Alter, Update, and Drop actions for the objects of Table type must work with the CreateInstance permission for the Project object.
- If the CreateInstance permission is not granted, the corresponding operations cannot be performed even though the mentioned permissions are granted. This is related to the internal implementation of MaxCompute. The Select permission for Table type objects must work with the CreateInstance permission. While performing cross-project operation, such as selecting the table of project B in the project A, you must have the project A CreateInstance and the project B Table select permissions.
- After a user or role is added, you must grant permissions to the user or role. MaxCompute authorization is an object-based authorization method. The permission data authorized by ACL is considered as a type of sub-resource of the object. Authorization can be performed only if the object exists. When the object is deleted, the authorized permission data is automatically deleted.

- **SQL92 Authorization**

  MaxCompute supports authorization using the syntax similar to the GRANT and REVOKE commands defined by SQL92. It grants or revokes permissions to/from the existing project object through simple authorization statements. The authorization syntax is as follows:

```
grant actions on object to subject
revoke actions on object from subject
actions ::= action_item1, action_item2, ...
object ::= project project_name | table schema_name |
           instance inst_name | function func_name |
            resource res_name
subject ::= user full_username | role role_name
```

  Users familiar with GRANT and REVOKE commands defined by SQL92 or with Oracle database security management can identify that the ACL authorization syntax of MaxCompute does not support [WITH GRANT OPTION] authorization parameters. For example, when User A authorizes User B to access an object, User B cannot grant the permission to User C. In this scenario, all permissions can be granted by one of the following three roles:

  - Project owner
  - Project administrator
  - Object creator

- **Use example of ACL authorization**

  In the following scenario, the Alibaba Cloud account user alice@aliyun.com is a newly added member to the project test_project_a, and Allen is a RAM-sub account added to bob@aliyun.

com. In test_project_a, they both must submit jobs, create tables, and view existing objects in the project.

The project administrator bob performs the following authorization operations:

```
    use test_project_a;
    add user aliyun$alice@aliyun.com;
    add user ram$bob@aliyun.com:Allen;
    create role worker;
    grant worker TO aliyun$alice@aliyun.com;
    grant worker TO ram$bob@aliyun.com:Allen;
    grant CreateInstance, CreateResource, CreateFunction, CreateTabl
e, List ON PROJECT test_project_a TO ROLE worker;
```

- **Cross-project Table/Resource/Function sharing**

Following the preceding example, aliyun$alice@aliyun.com and ram$bob@aliyun.com:Allen have certain permissions in test_project_a. These two users must query table prj_b_test_table in test_project_b, and use test_project_b. UDF prj_b_test_udf.

The project administrator performs the following authorization operations for test_project_b:

```
    use test_project_b; --Open the project
    add user aliyun$alice@aliyun.com; --Add the user
    add user ram$bob@aliyun.com:Allen; --Add th RAM sub-account
    create role prj_a_worker; --Create a role
    grant prj_a_worker TO aliyun$alice@aliyun.com; --Grant the role
    grant prj_a_worker TO ram$bob@aliyun.com:Alice; --Grant the role
    grant Describe , Select ON TABLE prj_b_test_table TO ROLE
prj_a_worker; --Authorize the role
    grant Read ON Function prj_b_test_udf TO ROLE prj_a_worker; --
Authorize the role
    grant Read ON Resource prj_b_test_udf_resource TO ROLE
prj_a_worker; --Authorize the role
--After authorization, the two users query table and use udf in
test_project_a as follows:
use test_project_a;
select test_project_b:prj_b_test_udf(arg0, arg1) as res from
test_project_b.prj_b_test_table;
```

Note:

If UDF is created in test_project_a, then only Resource authorization is required. Use the following code:

```
create function function_name as 'com.aliyun.odps.compiler.udf.
PlaybackJsonShrinkUdf' using 'test_project_b/resources/odps-compiler-
playback.jar' -f;.
```

8.7 Permission check

MaxCompute provides the ability to view multiple permissions, including the permissions of certain users or roles, and authorization lists of specified objects.

MaxCompute uses the markup characters A, C, D, and G when showing the permissions of users or roles. The meanings of these markup characters are as follows:

- A: Access allowed.
- D: Access denied.
- C: Access granted with conditions. It appears only in a policy authorization system.
- G: Access granted with conditions. Permission can be granted to objects.

An example of viewing permissions is as follows:

```
    odps@test_project> show grants for aliyun$odpstest1@aliyun.com;
    [roles]
    dev
    Authorization Type: ACL
    [role/dev]
    A projects/test_project/tables/t1: Select
    [user/odpstest1@aliyun.com]
    A projects/test_project: CreateTable | CreateInstance | CreateFunc
tion | List
    A projects/test_project/tables/t1: Describe | Select
    Authorization Type: Policy
    [role/dev]
    AC projects/test_project/tables/test_*: Describe
    DC projects/test_project/tables/alifinance_*: Select
    [user/odpstest1@aliyun.com]
    A projects/test_project: Create* | List
    AC projects/test_project/tables/alipay_*: Describe | Select
    Authorization Type: ObjectCreator
    AG projects/test_project/tables/t6: All
    AG projects/test_project/tables/t7: All
```

Note:

Currently, desc role only displays ACL information of project and table authorization types, while ACL of other objects (function, resource, instance, job) does not support display.

View permissions of a specified user

```
    show grants; --View permissions of the current user.
    show grants for <username>; --View access permissions of a
specified user. The operation can be executed by project owners and
administrators.
```

Example:

To view the user Alibaba Cloud account bob@aliyun.com permissions in the current project, run the following command on the client:

```
show grants for ALIYUN$bob@aliyun.com;
```

To view RAM sub-account permissions:

```
show grants for RAM$account:sub-account;
```

Example:

```
show grants for RAM$bob@aliyun.com:Alice;
```

View permissions of a specified role:

```
describe role --View access permissions granted to a specified role
```

Note:
In the public cloud environment, description role currently only displays ACL information of the object authorization type of project and table, while ACL information of other objects (such as function, resource, instance, job) is not displayed.

View the authorization list of a specified object:

```
show acl for [on type ];--View the user and role authorization list of
 a specified object
```

Note:
When *[on type <objectType>]* is excluded, the default type is Table.

8.8 Security configurations

MaxCompute is a multi-tenant data processing platform. Distinct tenants have distinct data security requirements. Therefore, MaxCompute provides project-level security configurations to comply with the unique requirements of individual tenants. Project owners can customize their external account support and authentication models.

MaxCompute provides multiple methods of orthogonal authorization, including Access Control List (ACL) authorization and implicit authorization. An object creator is automatically granted the object access permission. Not all users need these security features. Users can properly configure the project authentication model based on their service security requirements and usage patterns.

```
    show SecurityConfiguration
      --View the project security configuration.
    set CheckPermissionUsingACL=true/false
      --Enable/Disable the ACL authorization mechanism. The default
value is true.
    set ObjectCreatorHasAccessPermission=true/false
      --Enable/Disable automatic access permission granting to object
creators. The default value is true.
    set ObjectCreatorHasGrantPermission=true/false-* +
      --Enable/Disable automatic authorization permission granting to
object creators. The default value is true.
    set ProjectProtection=true/false
        --Enable/Disable project data protection to enable/disable
data transfer from the project.
```

Note:

You can also complete the security configuration of a project in a visualized technique using DataWorks. For more information, see *Project Management*.

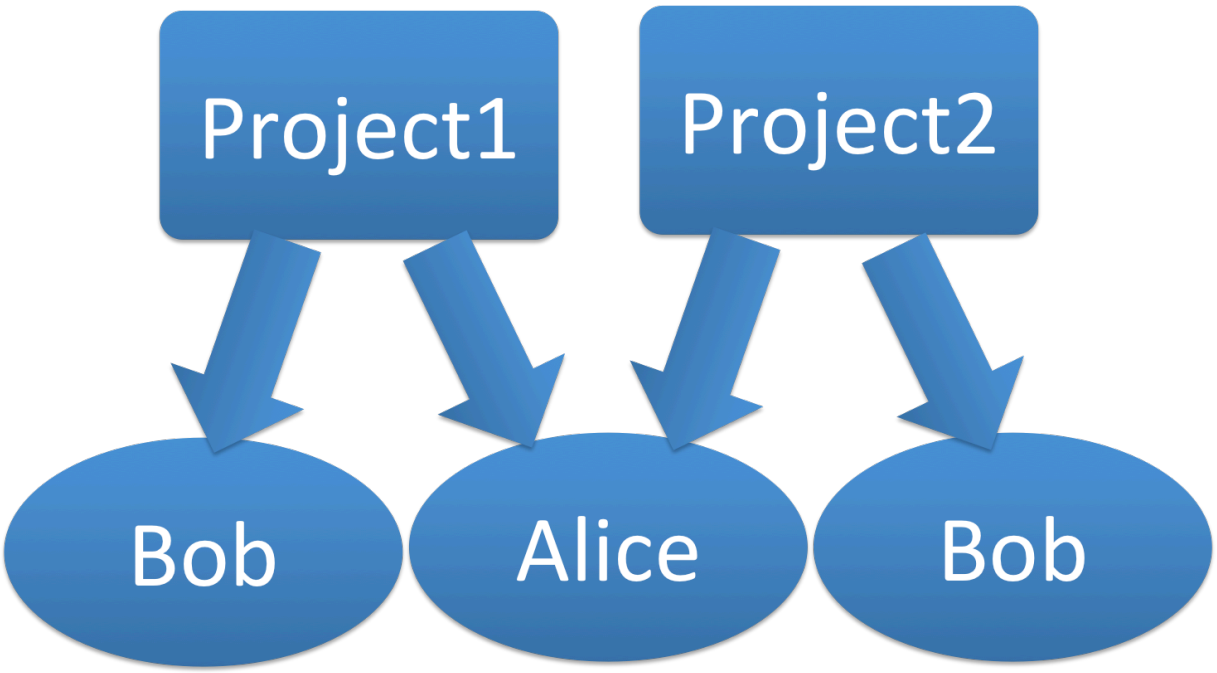
8.9 Data protection of projects

Background and motivation

Some companies (including financial institutions, military enterprises and so on) are extremely sensitive to data security. Hence, to secure the data, additional security measures are taken, that include not allowing employees to carry USB storage devices or personal hard disks to work; or most of the times the USB ports are disabled. Employees are not allowed to work from home. All these measures are taken to secure the sensitive data.

As a MaxCompute Project Space Administrator, do you have similar security requirements, where users are not allowed to move data out of the project space?

For example, when the owner of Project Space prj1 encounters this situation as shown in the following figure, are you worried that user Alice will transfer the data that she can access to prm9, only because she has access to prj2. prj2. and prj2?



More specifically, assume that Alice has been granted access to myprj, which is the Select permission for Table1, and then she is also granted create table permission by the administrator of prj2.

By these permissions, Alice is able to transfer the data to prm9 in any of the following ways:

- Submit SQL:

```
create table prj2.table2 as select * from myprj.table1;
```

- Write MapReduce to read myprj. Table1 and write to the scanner.

If the data in your project space is sensitive, you will be restricted to share data out of your project . MaxCompute can resolve issues pertaining to data protection and the aforementioned operations as well.

Data protection feature

MaxCompute provides a project space protection feature that helps to resolve issues mentioned earlier. As a user, set the project as follows:

```
    set projectProtection=true
```

```
      -- Set project protection rule: data can only flow and cannot
flow out
```

When project protection is set up, the data flow in your project space is controlled , "Data can only flow and cannot flow out ". That is, both of these actions will fail because they are against the project protection rule.

By default, ProjectProtection cannot be set and its value is false.

Also, users authorized to access multiple projects can freely use cross-project data access operations to share or transfer project data. If users are highly sensitive to project data security, the administrator must define a ProjectProtection feature likewise.

Data outflow method after enabling data protection

After setting ProjectProtection in the user's project, the user may soon make requests such as Alice applies to the user for exporting the data of a table out of the user’s project.

Moreover, user review confirms that this table does not contain sensitive data. In order not to affect Alice's normal business requirements, MaxCompute provides two data export methods to the user after setting ProjectProtection.

- **Set TrustedProject**

  In case, the current project space is protected, and if you set the target space for the data inflows to the trustedproject for the current space. Then, the data flow to the target project space will not be considered a violation of the project protection rule. If multiple project spaces are set to trustedproject between two and one another, so these project spaces form a trustedproject.

  Group; the data can flow within the project group, but restricted to be shared out of the project group.

  Use the following command to manage the TrustedProject:

  ```
  list trustedprojects;
    -- View All trustedprojects in the current project
  add trustedproject <projectname>;
    -- Add a trustdproject to the current project
  remove trustedproject <projectname>;
    -- Remove a trustdproject from the current project
  ```

- **Resource sharing and data protection**

  In MaxCompute, the *package-based resource sharing* feature and the project protection data protection feature are orthogonal, but they are similar to each other in terms of functions.

MaxCompute rules **give priority to resource sharing over data protection**. Therefore, if a data object allows access by users from other projects through resource sharing, the ProjectProtection rules will not apply to this data object.

Best practices

To prevent data outflow from the project, after setting `ProjectProtection=true`, check the following settings:

- Make sure the trustedproject is not added. If set, you must assess possible risks;
- Make sure that package data is not used for sharing. If set, make sure that no sensitive data exists in the package.

8.10 Security command list

8.10.1 Security configuration of a project

This article introduces you to the concept of authentication configuration and data protection in some project space security configurations.

Authentication configuration

Statement	Description
show SecurityConfiguration	View the security configuration of the project.
set CheckPermissionUsingACL=true/false	Enable/Disable the ACL-based authorization.
set CheckPermissionUsingPolicy=true/false	Enable/Disable the policy authorization.
set ObjectCreatorHasAccessPermission=true/false	Grant/Revoke default access permissions to/from object creators.
set ObjectCreatorHasGrantPermission=true/false	Grant/Revoke default authorization permissions to/from object creators.

Data protection

Statement	Description
set ProjectProtection=false	Disable data protection.
list TrustedProjects	View the list of trusted projects.
add TrustedProject <projectName> `<projectName>`	Add a trusted project.
remove TrustedProject `<projectName>`	Remove a trusted project.

8.10.2 Manage permissions

This article introduces you to the related concepts of user management, role management, ACL authorization, and permission review in project space rights management.

Manage users

Statement	Description
list users	View all users added to the project.
add user <username> `<username>`	Add a user.
remove user <username> `<username>`	Remove the user.

Manage roles

Statement	Description
list roles	View all created roles.
create role <rolename> `<rolename>`	Create a role.
drop role <rolename> `<rolename>`	Delete a role.
grant `<rolelist>` to `<username>`	Assign one or multiple roles to the user.
revoke `<rolelist>` from `<username>`	Revoke a role from the user.

ACL Authorization

Statement	Description
grant `<privList>` on `<objType>` `<objName>` to user `<username>`	Authorize a user.
grant `<privList>` on `<objType>` `<objName>` to role `<rolename>`	Authorize a role.
revoke `<privList>` on `<objType>` `<objName>` from user `<username>`	Revoke user authorization.
revoke `<privList>` on `<objType>` `<objName>` from role `<rolename>`	Revoke role authorization.

Permission review

Statement	Description
whoami	View current user information.
show grants [for `<username>`] [on type `<objectType>`]	View user role and permissions.

Statement	Description
show acl for <objectName> [on type <objectType>]	View specific object authorization information.
describe role <roleName>	View role authorization information and role assignments.

8.10.3 Package-based resource sharing

This article gives you a description of resource sharing statements based on Package.

Share resources

Statement	Description
Create package <pkgname> <pkgName>	Create a package.
Delete package <pkgname> <pkgName>	Delete a package.
add <objType> <objName> to package <pkgName> [with privileges privs]	Add resources to be shared to a package.
remove <objType> <objName> from package <pkgName>	Remove shared resources from a package.
allow prOject <prjName> to install package <pkgName> [using label <num>]	Allow a project to use a user package.
disallow project <prjName> to install package <pkgName>	Disallow a project from using a user package.

Use Resources

Statement	Description:
Install package <pkgname> <pkgName>	Install a package.
uninstall package <pkgName>	Uninstall a package.

View a package

Statement	Description:
show packages	List all created and installed packages.
describe package <pkgName>	View details of a package.

Note:

If you execute a production plan authorization related request in DataWorks：

1. Project owner is executed by temporary query and cannot be submitted to the production environment for execution.Because the production environment is executed by the production account, which has no authorized authority.
2. Add the **use<production project>;** statement before the query and submit it with the command. Because DataWorks data development defaults the current project is the development project ending in _dev. When executing authorization commands from the command line, ask project owner to execute them first:

```
use project_name;--进入对应的project
```

8.11 Resource share across project space

8.11.1 Resource sharing across projects based on package

Assume that you are the project owner or administrator (admin role) of a few projects. One of your primary accounts has multiple projects, wherein the project prj1 has some resources (including tables, resources, and custom functions) that can be shared with other projects. However, adding users of other projects to prj1 and granting permissions to them one by one is complicated, and adding the users who are irrelevant but are added to the prj1 project (if they exist) complicates the project management.This section describes cross-project resource sharing.

If resources must be controlled by the user in a fine-grained manner, and the user who applies for the control permission is a member of the business project team, we recommend using the *Project user and authorization management* feature.

Package is used for sharing data and resources across projects. It solves the problem of cross-project user authorization.

Use package to solve the following problems effectively:

If members of the Alifinance project want to access data in the Alipay project, the administrator of the Alipay project must perform tedious authentication operations: First, add users in the Alifinance project to the Alipay project, and then perform general authentications on the newly added users, respectively.

Actually, the administrator of the Alipay project does not want to authenticate and manage all users in the Alifiance project. Instead, the administrator expects more efficient feature for autonomous authentication controls over permissive objects.

After Package is used, the administrator of the Alipay project can perform packaging authorization on the objects to be used by the Alifinance project (that is, create a Package), and then permit the Alifinance project to install the Package. After the Alifinance project's administrator installs the Package, the administrator can determine whether to grant permissions of the Package to the users of the Alifinance project as required.

8.11.2 Package usage method

This article introduces you to the operations involved in the project space Package creator and Package consumer.

Package usage method

The use of package involves two subjects: the package creator and the package user.

- The package creator provides the resources to be shared and the permissions to access it. It also allows the package user to install and use it.
- The package user uses the package. After the package is published, the user can directly access the resource across projects.

The following is a description of the operations involved with the package creator and package user.

Package creator

- Create package

```
create package <pkgname>;
```

Note:

- Only the project owner has the permission to create a package.
- The name of the package cannot exceed 128 characters.

- Add a resource to be shared to the package

```
    Add project_object to package package_name [with privileges] --
add objects to package
    Remove project_object from package package_name; -- remove
object from package
    project_object ::= table table_name |
                       instance inst_name |
                       function func_name |
                       resource res_name
    privileges ::= action_item1, action_item2, ...
```

Note:

- Currently, supported types of objects exclude projects. Therefore, you cannot use a package to create objects in other projects.
- The objects themselves and the permission to perform operations on them are added to the package at the same time. When not passed (with privileges) even specifying an action permission, the default is read-only, that is, read/describe/select. The object and its permissions are treated as a whole and cannot be updated once added. If necessary, you can only delete and re-add.
- When an object is added to a package, it is not packaged as a snapshot, so subsequent object data changes, and access to the object through package authorization is also the current data of the object.

Use the following commands to perform various operations on the package:

- Allow other projects to use a package

```
allow project <prjName> to install package <pkgName> [using label <
num>]
```

- Revoke other projects' permission to use a package

```
disallow project <prjName> to install package <pkgName>
```

- Drop a package

```
Delete package <pkgname>;
```

- View the list of packages already created and installed

```
Show packages;
```

- View package details

```
Describe package <pkgname>;
```

Package users

- Install package

```
Install package <pkgname>;
```

For package installation, the pkgName format is: <projectName>.<packageName>.

Note:
Only the project owner has permissions to perform this operation.

- Uninstalling package

```
Uninstall package <pkgname>;
```

For package installation, the pkgName format is: <projectName>.<packageName>.<projectName>.<packageName>

- View a package

```
    Show packages;
    View the list of packages already created and installed
    Describe package <pkgname>;
    View details of package
```

- Client project grants access to package to other members or role of this project

The installed package is an independent type of MaxCompute object. To access resources in a package (resources shared with you by other projects), you must have the permission to read package.

If you do not have the Read permission, you must apply to the project owner or admin for the permission. The project owner or admin can grant permissions through ACL authorization or policy authorization.

Authorize package to user or role:

```
grant actions on package <pkgName> to user <username>;
grant actions on package <pkgName> to role <role_name>;
```

Note:
After authorization, user has access to the object in that package only in this project.

For example, the following ACL authorization allows the cloud account user odps_test@aliyun.com to access resources in the package:

```
    use prj2;
    install package prj1.testpkg;
    grant read on package prj1.testpackage to user aliyun$odps_test@
aliyun.com;
```

]

Or allow all members of role role_dev to access resources in package:

```
use prj2;
    install package prj1.testpkg;
```

```
    grant read on package prj1.testpackage to role role_dev;
```

Example

Jack is the administrator of prj1. John is the administrator of prj2. To address some business needs, Jack wants to share some resources of prj1 (such as datamining.jar and sampletable) to John's prj2. If prj2 user Bob must access these resources, the prj2 administrator can self-authorize Bob through ACL administrator or policy authorization without Jack’s involvement.

Procedure:

1. Prj1 administrator Jack creates resources package in prj1.

```
    Use prj1;
    Create package datamicing; -- creating a package
    Add Resource dating. jar to package dating;-add resource to
package
    Add Table sampletable to package dating; -- adding table to
package
    Allow project prm9 to install package dating; -- sharing package
 to Project Space prm9
```

2. Prj2 administrator Bob installs a package in prj2.

```
    use prj2;
    install package prj1.datamining; -- installs a package
    describe package prj1.datamining; -- view a list of resources in
 the package
```

3. Bob self-authorizes the package.

```
    use prj2;
    grant Read on package prj1.datamining to user aliyun$bob@aliyun.
com; -- authorization of Bob to use package via ACL
```

8.12 Column-level access control

Label-based security (LabelSecurity) is a required MaxCompute Access Control (MAC) policy at the project space level. It allows project administrators to control the user access to column-level sensitive data with improved flexibility.

Difference between MAC and DAC in MaxCompute

In MaxCompute, MAC is independent of Discretionary Access Control (DAC). Two examples are provided to illustrate the differences between MAC and DAC.

To drive a vehicle, you must first have to apply and acquire a valid driver's license, similarly, a user who wants to read data in a MaxCompute project must first apply for the SELECT permission. The permission application is within the scope of DAC.

Because the country with a high accident rate, drunk driving is strictly restricted. To curb this , all drivers are required to have a driver's license and must not drink and drive. Likewise, in MaxCompute, reading highly sensitive data is analogous to the law against drunk driving. The read prohibition is within the scope of MAC.

Data sensitivity classification

LabelSecurity assigns security levels to data and the users who access the data. In the government and financial sectors, data sensitivity is usually classified into four levels: 0 (Unclassified), 1 (Confidential), 2 (Sensitive), and 3 (Highly Sensitive). MaxCompute adopts such classification. Project owners must define standards for data sensitivity classification and access level classification. The default access level of all users is 0, and the default sensitivity level of data is 0.

LabelSecurity supports data sensitivity classification at the column level. Administrators can set sensitivity labels for all the columns of a table. A table may have columns of different sensitivity levels.

Administrators can also set sensitivity labels for views. A view and its base table have independent sensitivity labels. The default sensitivity level of a new view is 0.

Default security policies of LabelSecurity

LabelSecurity applies the following default security policies to the data and users assigned with sensitivity or security labels:

- No-ReadUp: A user is not allowed to read data with a sensitivity level higher than the user level unless the user is explicitly authorized.
- Trusted-User: A user is allowed to write data of all sensitivity levels. The default sensitivity level of new data is 0 (unclassified).

Note:

- In some traditional MAC systems, other complex security policies are applied to prohibit unauthorized data distribution in a project. For example, the No-WriteDown policy prohibits users from writing data with a sensitivity level not higher than the user level. By default, MaxCompute does not support No-WriteDown, considering the costs involved in managing the data sensitivity levels of project administrators. The effect of No-WriteDown can be attained by modifying the project security settings (`Set ObjectCreatorHasGrantPermission=false`).

- To prohibit data flowing among different projects, you can set the projects to the protected state (ProjectProtection). With the setting, users can only access the data within their projects . This prevents data transfer or data sharing outside the project.

By default, projects disable LabelSecurity. The project owners can enable it as required.

After LabelSecurity is enabled, the default security policies are executed. When a user accesses a data table, the user must have the SELECT permission and the access level required for sensitive data reading. Compliance with LabelSecurity is a required but not the sufficient condition for passing CheckPermission.

LabelSecurity operations

- **Enable or disable LabelSecurity**

```
 Set LabelSecurity=true|false;
   -- Enables or disables LabelSecurity. The default value is false.
   -- LabelSecurity can be enabled or disabled only by the project
owner. Other operations can be performed by the project administra
tor.
```

- **Set security labels for users**

```
 SET LABEL <number> TO USER <username>;-- Value range of "number": [
0, 9]. This operation can be performed only by the project owner or
administrator.
 -Example:
 ADD USER aliyun$yunma@aliyun.com; --Adds a user with the default
security label 0.
 ADD USER ram$yunma@aliyun.com:Allen; --Adds user Allen, which is a
RAM subaccount of yunma@aliyun.com.
 SET LABEL 3 TO USER aliyun$yunma@aliyun.com;
   -- Sets the security label of yunma to 3 to allow this user to
access only the data with a sensitivity level not higher than 3.
 SET LABEL 1 TO USER ram$yunma@aliyun.com:Allen;
   -- Sets the security label of subaccount Allen to 1 to allow this
 user to access only the data with a sensitivity level not higher
than 1.
```

- **Set sensitivity labels for data**

```
 SET LABEL <number> TO TABLE tablename(column_list); -- Value range
 of "number": [0, 9]. This operation can be performed only by the
project owner or administrator.
 -Example:
 SET LABEL 1 TO TABLE t1;  --Sets the sensitivity label of table t1
to 1.
 SET LABEL 2 TO TABLE t1(mobile, addr); --Sets the sensitivity
labels of the "mobile" and "addr" columns of table t1 to 2.
 SET LABEL 3 TO TABLE t1;  --Sets the sensitivity label of table t1
to 3.  The sensitivity labels of the "mobile" and "addr" columns are
 still 2.
```

Note:

The sensitivity labels explicitly set for the columns overwrite the sensitivity label set for the table, without considering the label setting order and the sensitivity level.

- **Explicitly authorize lower-level users to access specific data tables with a high sensitivity level**

```
 --Grant permissions:
 GRANT LABEL <number> ON TABLE <tablename>[(column_list)] TO USER <
username> [WITH EXP <days>]; --The default validity period is 180
days.
 -- Revoke the permissions:
 REVOKE LABEL ON TABLE <tablename>[(column_list)] FROM USER <
username>;
 -- Clear the expired permissions:
 CLEAR EXPIRED GRANTS;
 -Example:
 GRANT LABEL 2 ON TABLE t1 TO USER ram$yunma@aliyun.com:Allen WITH
EXP 1; --Explicitly authorizes Allen to access the data of table t1
with a sensitivity level not higher than 2 for a period of 1 day.
 GRANT LABEL 3 ON TABLE t1(col1, col2) TO USER ram$yunma@aliyun.com
:Allen WITH EXP 1; --Explicitly authorizes Allen to access the data
 in col1 and col2 of table t1 with a sensitivity level not higher
than 3 for a period of 1 day.
 REVOKE LABEL ON TABLE t1 FROM USER ram$yunma@aliyun.com:Allen; --
Revokes the permission of Allen to access the sensitive data in
table t1.
```

Note:

Once the label-authorized permission of a user to access a table is revoked, the permission to access the table fields of the same user is also revoked.

- **List the sensitive data sets that a user can access**

```
 SHOW LABEL [<level>] GRANTS [FOR USER <username>];
    --When [FOR USER <username>] is unspecified, the system lists
the sensitive data sets that the current user can access.
   --When <level> is unspecified, the system lists the permissions
 granted by all label levels.When <level> is specified, the system
lists only the permissions granted by a specific label level.
```

- **List the users who can access a specific table containing sensitive data**

```
 SHOW LABEL [<level>] GRANTS ON TABLE <tablename>;
   --Displays the label-authorized permissions on the specified
table.
```

- **List the label-authorized permissions of a user at all levels to access a data table**

```
 SHOW LABEL [<level>] GRANTS ON TABLE <tablename> FOR USER <username
>;
```

```
   --Displays the label-authorized permissions of the specified user
 to access the columns of a specific table.
```

- **List the sensitivity levels of all the columns of a table**

```
DESCRIBE <tablename>;
```

- **Control the access level of a package installer regarding the sensitive resources of the package**

```
 ALLOW PROJECT <prjName> TO INSTALL PACKAGE <pkgName> [USING LABEL <
number>];
   --The package creator grants an access level to the package
installer regarding the sensitive resources of the package.
```

Note:

- When `[USING LABEL <number>]` is unspecified, the default access level is 0. The package installer can only access non-sensitive data.
- When accessing to sensitive data across projects, the access level defined by this command applies to all the users in the project of the package installer.

LabelSecurity use cases

- **Prohibit all the users in a project except the project administrator from reading some sensitive columns of a table**

  Description:

  user_profile is a table with sensitive data in a project. It has 100 columns, five of which contain sensitive data: id_card, credit_card, mobile, user_addr, and birthday. DAC grants all users the SELECT permission on this table. The project owner wants to prohibit all the project users except the project administrator from reading the sensitive columns of the table.

  To achieve this purpose, the project owner can perform the following operations:

```
set LabelSecurity=true;
  --Enables LabelSecurity.
set label 2 to table user_profile(mobile, user_addr, birthday);
  --Sets the sensitivity level of the specified columns to 2.
set label 3 to table user_profile(id_card, credit_card);
  --Sets the sensitivity level of the specified columns to 3.
```

Note:

After the preceding operations, non-administrator users cannot access the data in the five columns. To access the sensitive data for business purposes, the user must be authorized by the project owner or administrator.

Solution:

Alice is a member of the project. For official purposes, she wants to apply for access to the data in the mobile column of table user_profile for a period of one week. To authorize Alice, the project administrator can perform the following operation:

```
GRANT LABEL 2 ON TABLE user_profile TO USER ALIYUN$alice@aliyun.com
WITH EXP 7;
```

Note:

Mobile, user_addr, and birthday column contain data with a sensitivity level of 2. Birthday. After authorization, Alice can access the data in these three columns. The authorization causes the issue of excessive permission grants. This issue can be avoided if the project administrator sets the sensitive columns properly.

- **Prohibit the project users with access to sensitive data from copying and distributing the sensitive data within the project without authorization**

  Description:

  In the preceding use case, Alice is granted the access permission on the data with a sensitivity level of 2 for official purposes. The project administrator worries that Alice may copy that data from table user_profile to table user_profile_copy created by her and grants Bob the access permission on user_profile_copy. The project administrator needs a method to restrict Alice's actions.

  Solution:

  Considering security usability and management costs, LabelSecurity adopts the default security policy that allows for WriteDown. Users can write data to the columns with a sensitivity level not higher than the user level. MaxCompute cannot address the preceding requirement of the project administrator. However, the project administrator can restrict the discretionary authorization behavior of Alice by allowing her to only access the data she created, but disallowing her to grant the data access permission to other users. The procedure is as follows:

```
 SET ObjectCreatorHasAccessPermission=true;
   --Allows the object creator to operate objects.
 SET ObjectCreatorHasGrantPermission=false;
   --Prohibits the object creator from granting the object access
permission to other users.
```

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.

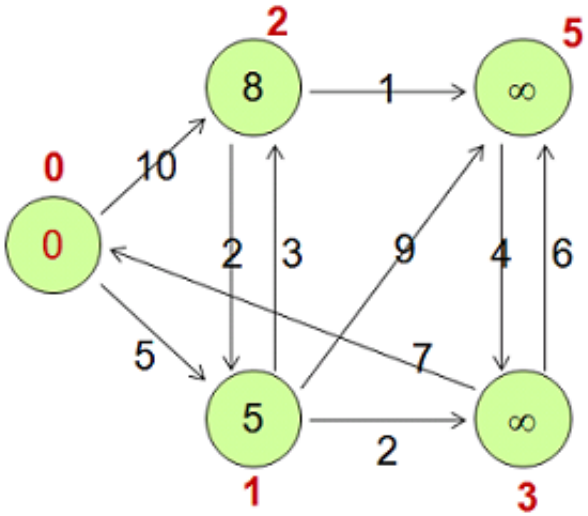
This process outputs a final solution after performing iterative graph editing and evolution. Typical applications include *PageRank*, *SSSP algorithm*, and *Kmeans algorithm*. Use Java SDK, an interface provided by MaxCompute Graph to compile graph computing programs.

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:

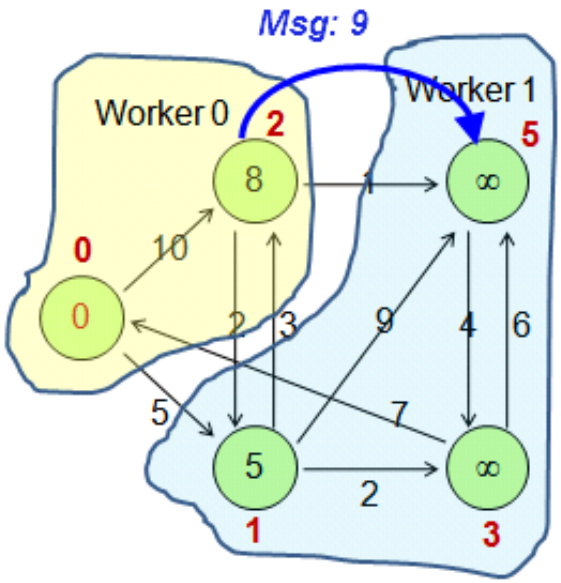
Vertex	<ID, Value, Halted, Edges>
v0	<0, 0, false, [<1, 5 >, <2, 10 >] >
v1	<1, 5, false, [<2, 3>, <3, 2>, <5, 9>]>
v2	<2, 8, false, [<1, 2>, <5, 1 >]>
v3	<3, Long.MAX_VALUE, false, [<0, 7>, <5, 6>]>
v5	<5, Long.MAX_VALUE, false, [<3, 4 >]>

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.

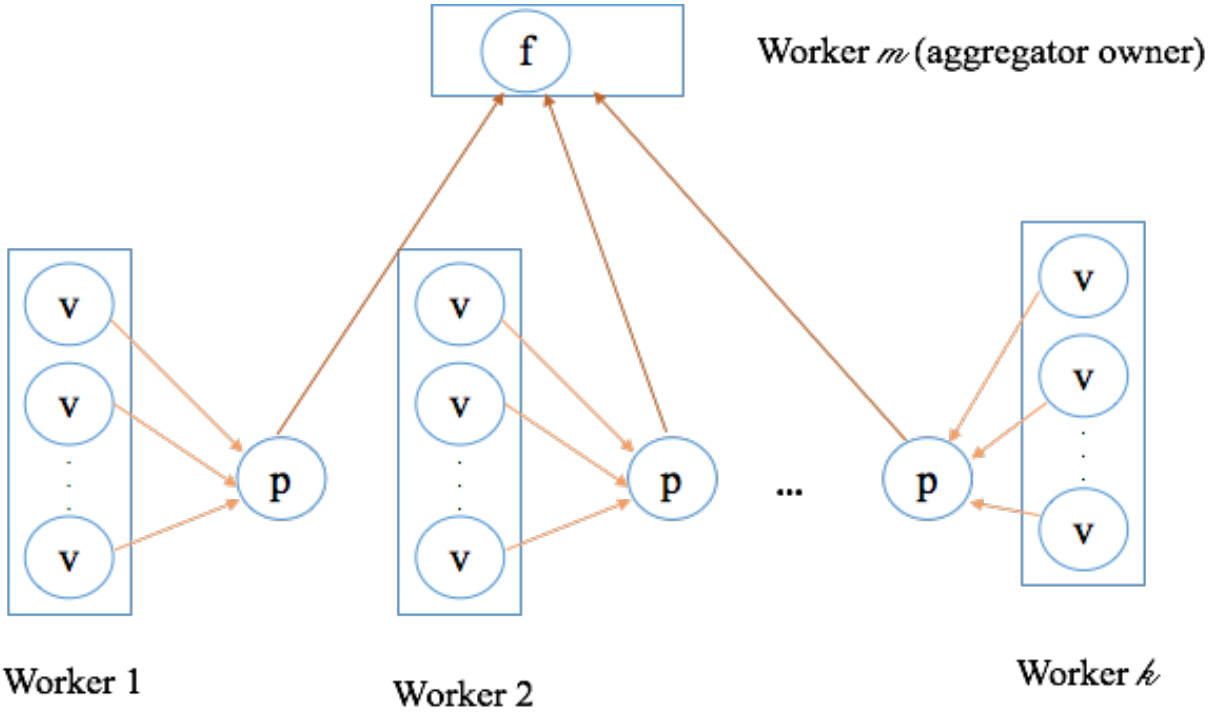
In MaxCompute Graph, Aggregator helps to collect and process global information. In MaxCompute Graph, Aggregator is used to summarize and process global information.

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:

  1. merge(p1, p0) runs first, and p1 and p0 are aggregated as p1'.
  2. 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 {
    wirteForDenseVector(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*. wirteForDenseVector() 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, KmeansValu
e, 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, NullWritab
le> 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 {
    wirteForDenseDenseMatrix(out, centroids);
    wirteForDenseDenseMatrix(out, sums);
    wirteForDenseVector(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 aggregatio n.

The next is KmeansAggregator used for custom Aggregator implementation. The following describes implementation in order of the preceding APIs.

1. Run createStartupValue(), see the following:

```
public static class KmeansAggregator extends Aggregator<KmeansAggr
Value> {
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.

2. 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 newCentriods = calculateNewCentroids(value.sums, value.
counts, value.centroids);
 // print old centroids and new centroids for debugging
 System.out.println("\nsuperstep: " + context.getSuperstep() +
     "\nold centriod:\n" + value.centroids + " new centriod:\n" +
newCentriods);
 boolean converged = isConverged(newCentriods, 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 centriods
   for (int i = 0; i < newCentriods.numRows(); i++) {
     Writable[] centriod = new Writable[newCentriods.numColumns()];
     for (int j = 0; j < newCentriods.numColumns(); j++) {
       centriod[j] = new DoubleWritable(newCentriods.get(i, j));

     context.write(centriod);

   // true means to terminate iteration
   return true;

 // update centriods
```

```
value.centroids.set(newCentriods);
// 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 aggregatio n result and determines whether to end the iteration.

Attachment

Kmeans

9.3 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 as 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 OVERWRITE. 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 IOException;
                                    public Iterable<byte[]>
readCacheArchive(String resourceName, String relativePath)throws
IOException;
                                    public Iterable<WritableRecord>
readResourceTable(String resourceName);
public BufferedInputStream readCacheFileAsStream(String resourceName
) throws IOException;
public Iterable<BufferedInputStream> readCacheArchiveAsStream(String
 resourceName) throws IOException;
public Iterable<BufferedInputStream> readCacheArchiveAsStream(String
 resourceName, String relativePath) 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*.

9.4 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.
Vertex	A vertex is a node that is defined by the attributes including ID, value , halted, and edges. A vertex is implemented by the setVertexClass interface of GraphJob.

Main interface	Description
Edge	Edge is the abstract of edges in a graph, including the attributes destVertexId and value. Adjacent tables are used as the graph data structure, and outbound edges of a vertex are stored in edges of the vertex.
GraphLoader	GraphLoader is used to load graphs. GraphLoader is implemented by using the setGraphLoaderClass interface of GraphJob.
VertexResolver	VertexResolver is used to customize the conflict processing logic to modify graph topology. The setLoadingVertexResolverClass and setComputingVertexResolverClass interfaces of GraphJob provide the conflict processing logic for graph topology modification during graph loading and iteration calculation.
Partitioner	Partitioner is used to partition a graph so that the calculations can be fragmented. Partitioner is implemented by using the setPartitionerClass interface of GraphJob. HashPartitioner 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	WorkerComputer allows a Worker to run a custom logic during startup and exit. WorkerComputer is implemented by using the setWorkerC omputerClass interface of a GraphJob.
Aggregator	setAggregatorClass(Class …) defines one or multiple Aggregators.
Combiner	setCombinerClass sets a Combiner.
Counters	Indicates a counter. In job running logic, the WorkerContext 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.

9.5 Development and debugging

MaxCompute does not provide Graph development plugins for users. However, you can develop the MaxCompute Graph program based on Eclipse. The development process is as follows:

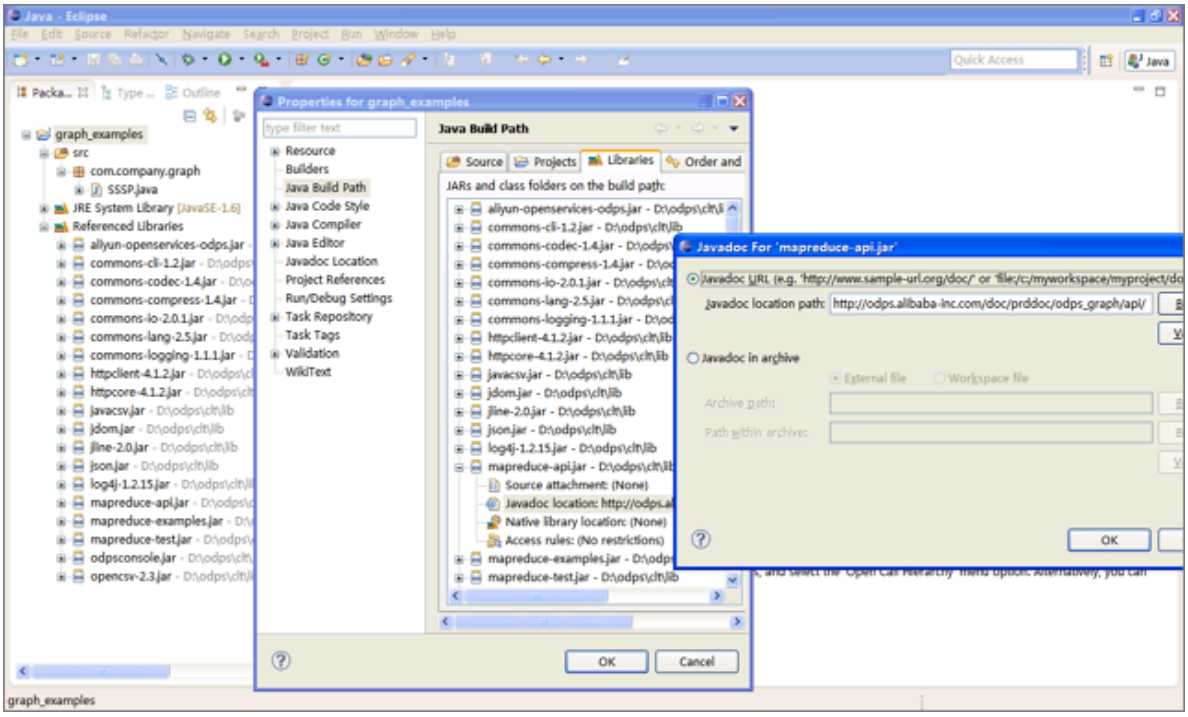
1. Compile Graph codes and perform basic tests using local debugging.
2. Perform cluster debugging and verify the result.

Example

This section uses the *SSSP* algorithm as an example to describe how to use Eclipse to develop and debug a Graph program.

Procedure

1. Create a Java project, for example, graph_examples.
2. Add the JAR package in the lib directory of the MaxCompute client to Build Path of the Eclipse project. The following figure shows a configured Eclipse project:



3. Develop a MaxCompute Graph program.

   In the actual development process, an example (such as *SSSP*) is often copied and then modified. In this example, only the package path is changed to package com.aliyun.odps.graph.example.

4. Compile and build the package.

   In an Eclipse environment, right-click the source code directory (the src directory in the figure) and select Export > Java > JAR file to generate a JAR package. Select the path for storing the target JAR package, for example, `D:\\odps\\clt\\odps-graph-example-sssp.jar`.

5. Use the MaxCompute console to run SSSP. For more information about the related operations, see *Run Graph* in “Quick start”.

Note:

For more information about the related development procedure, see Introduction on the *Graph development plug-in*.

Local debugging

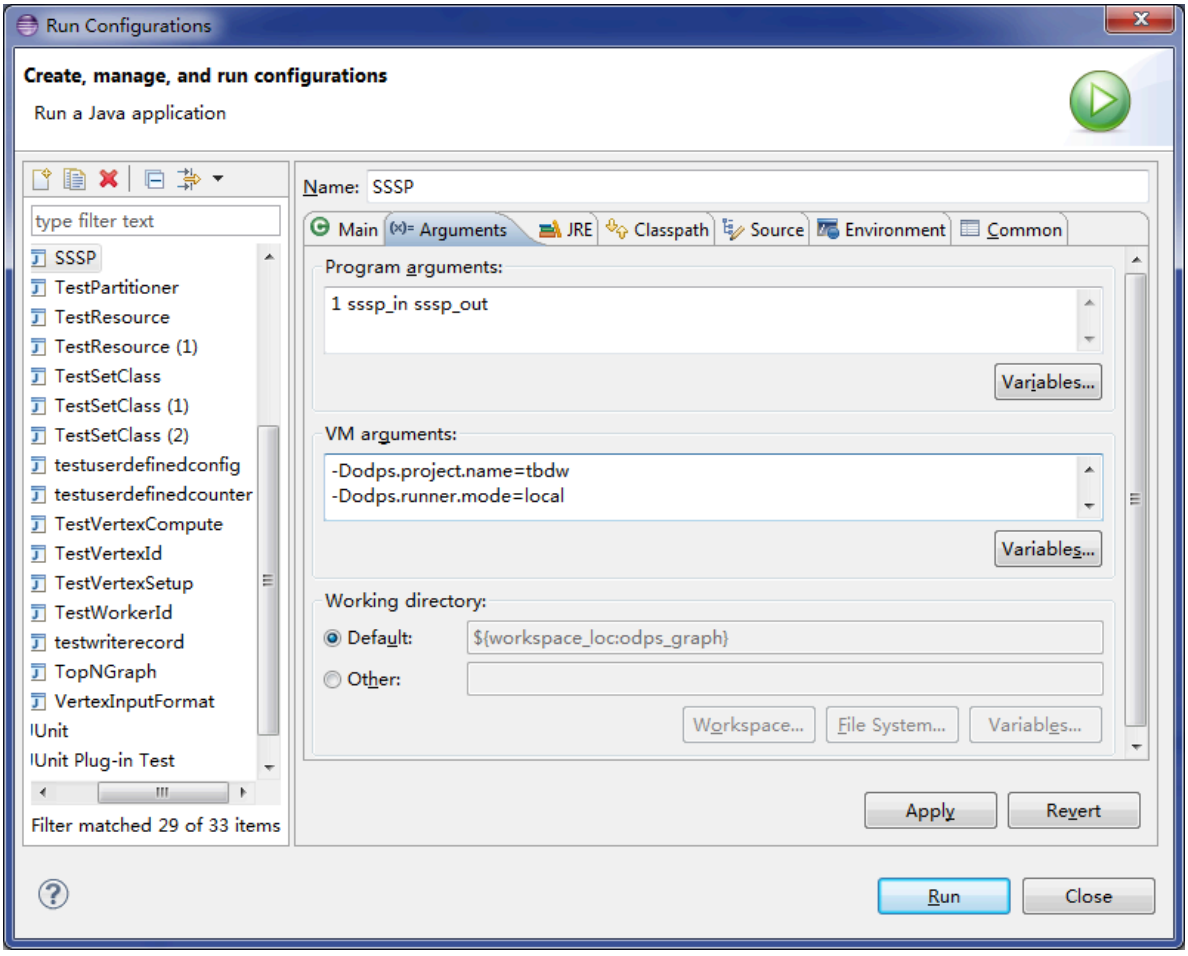
MaxCompute Graph supports the local debugging mode. Use Eclipse to perform breakpoint debugging.

Procedure

1. Download an odps-graph-local maven package.
2. Select the Eclipse project, right-click the main program file (including the main function) of the Graph job, and configure its running parameters (by selecting Run As > Run Configurations).
3. On the Arguments tab page, set Program arguments to `1 sssp_in sssp_out` as the input parameter of the main program.
4. On the Arguments tab page, set VM arguments to the following:

```
-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. If MapReduce is in local mode (the value of odps.end.point is not specified), you must create the sssp_in and sssp_out tables in the warehouse and add data for sssp_in. Input data is listed as follows:

```
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, see *MapReduce local running*.

6. Click **Run**.

Note:

Check the settings of conf/odps_config.ini in the MaxCompute client to set parameters. The preceding parameters are commonly used. Other parameters are described as follows:

- odps.runner.mode: The parameter value is local. This parameter is required for the local debugging function.

- odps.project.name: (Required). Specifies the current project.
- odps.end.point: (Optional). Specifies the address of the current MaxCompute service. If this parameter is not specified, metadata of tables or resources is only read from the warehouse, and an exception is thrown when the address does not exist. If this parameter is specified, data is read from the warehouse first, and then from remote MaxCompute if the address does not exist.
- odps.access.id: Indicates the ID to connect to the MaxCompute service. This parameter is valid only when odps.end.point is specified.
- odps.access.key: Indicates the key to connect to the MaxCompute service. This parameter is valid only when odps.end.point is specified.
- odps.cache.resources: Specifies the resource list in use. This parameter has the same effect as -resources of the JAR command.
- odps.local.warehouse: Specifies the local warehouse path. This parameter is set to ./warehouse by default ,if not specified.

After SSSP debugging is implemented locally in Eclipse, the following information is output:

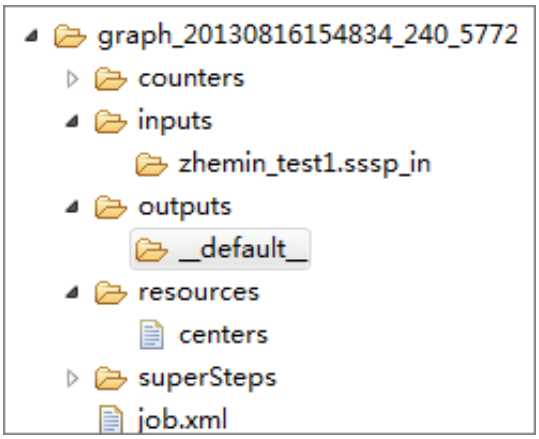
```
Counters: 3
        com.aliyun.odps.graph.local.COUNTER
                TASK_INPUT_BYTE=211
                TASK_INPUT_RECORD=5
                TASK_OUTPUT_BYTE=161
                TASK_OUTPUT_RECORD=5
 graph task finish
```

Note:

In the preceding example, the sssp_in and sssp_out tables must exist in the local warehouse. For more information about the sssp_in and sssp_out tables, see *Run Graph* in “Quick start”.

Temporary directory of local job

A temporary directory is created in the Eclipse project directory when local debugging runs each time, as shown in the following figure.



The temporary directory of a locally running Graph job contains the following directories and files:

- counters: Stores counting information about job running.
- inputs: Stores input data of the job. Data is preferentially obtained from the local warehouse. If such data does not exist locally, the MaxCompute SDK reads data from the server (if odps.end.point is set). An input reads only 10 data records by default. This threshold can be modified in the `-Dodps.mapred.local.record.limit` parameter, of which the maximum value is 10,000.
- outputs: Stores output data of the job. If the local warehouse has an output table, the result data in the outputoverwrites the corresponding table in the local warehouse after job running is complete.
- resources: Stores resources used by the job. Similar to inputs, data is preferentially obtained from the local warehouse. If such data does not exist locally, the data is read from the server using MaxCompute SDK (when odps.end.point is set).
- job.xml: Indicates job configuration.
- superstep: Stores information about message persistence in each iteration.

Note:
If a detailed log must be output during local debugging, the following log4j configuration file must be placed in the src directory: `log4j.properties_odps_graph_cluster_debug`.

Cluster debugging

After local debugging, 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 run the job, and view the running log and result data.

Note:

For more information about how to run Graph in a cluster, see *Run Graph* in “Quick start”.

Performance Tuning

The following section describes common performance tuning methods on the MaxCompute Graph framework.

Job Parameter configuration

GraphJob configurations that have an impact on performance include:

- setSplitSize(long): Indicates the split size of an input table. The unit is in MB. Its value must be greater than 0, and the default value is 64.
- setNumWorkers(int): Specifies the number of Workers for a job. The value range is [1, 1000], and the default value is –1. The number of Workers varies depending on the number of input bytes of the job and split size.
- setWorkerCPU(int): Indicates CPU resources of the Map. A one-core CPU contains 100 resources. The value range is [50, 800], and the default value is 200.
- setWorkerMemory(int): Indicates memory resources of the Map. The unit is MB. The value range is [256 MB, 12 GB], and the default value is 4,096 MB.
- setMaxIteration(int): Specifies the maximum number of iterations. The default value is –1. If the value is smaller than or equal to 0, the maximum number of iterations is not a condition for job termination.
- setJobPriority(int): Specifies the job priority. The value range is [0, 9], and the default value is 9 . A larger value indicates a smaller priority.

Additional actions that increase overall processing capabilities are as follows:

- You can use the setNumWorkers() method to increase the number of Workers.
- You can use the setSplitSize() method to reduce the split size and increase the speed for a job to load data.
- Increase the CPU or memory of Workers.
- Set the maximum number of iterations. If applications do not have high requirements on result precision, you can reduce the number of iterations to speed up the process.

The interfaces setNumWorkers and setSplitSize can be used together to speed up data loading. Assume that setNumWorkers is workerNum and setSplitSize is splitSize, and the total

number of input bytes is inputSize. The number of splits is calculated using the formula: splitNum = inputSize /splitSize. The relationship between workerNum and splitNum is as follows:

- If splitNum == workerNum, each Worker is responsible for loading one split.
- If splitNum > workerNum, each Worker is responsible for loading one or multiple splits.
- If splitNum < workerNum, each Worker is responsible for loading zero or one split.

Therefore, if the first two conditions are met, you can adjust workerNum and splitSize to enable fast data loading. In the iteration phase, you only need to adjust workerNum.

If you set runtime partitioning to false, we recommend that you use setSplitSize to control the number of Workers. Regarding the third condition, the number of vertices on some Worker may be 0. You can use set odps.graph.split.size=<m>; set odps.graph.worker.num=<n>; before the JAR command, which has the same effect as setNumWorkers and setSplitSize.

Another common performance problem is data skew. For example, on Counters, the number of vertices or edges processed by some Workers is much greater than that processed by other Workers.

Data skew occurs usually when the number of vertices, edges, or messages corresponding to some keys is much greater than that corresponding to other keys. Such keys with the large data volume are processed by a small number of Workers, resulting in a long run time of these Workers.

To resolve this problem, we recommend the following steps:

- Use a combiner to locally aggregate messages of vertices corresponding to such keys to reduce the number of sent messages.
- Improve the service logic.

Use a Combiner

Define a Combiner to reduce memory that stores messages and network data traffic volume and shortens the job execution time. For more information, see introduction to Combiner in MaxCompute SDK.

Reduce the Data Input Volume

When the data volume is large, reading data in a disk may extend the processing time. Therefore, reducing the number of data bytes to be read can increase the overall throughput, thereby improving job performance. You can use either of the following methods:

- Reduce the input data volume: For decision-making applications, results obtained from processing subsets after data sampling only affect the result precision, instead of the overall accuracy. Therefore, 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 (transmitted using column name arrays), rather than reading the entire table or table partition. This reduces the input data volume and improves job performance.

Built-in JAR Packages

The following JAR packages are loaded to JVMs running the Graph program by default. You do not have to upload these resources or carry these JAR packages when running -libjars on the command line.

- 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 a classpath that runs a JVM, the preceding built-in JAR packages are placed before users' JAR packages, which may result in a version conflict. For example, if your program uses a function of a class in commons-codec-1.5.jar but this function is not in commons-codec-1.3.jar. Check whether an implementation method exists in commons-codec-1.3.jar or wait for MaxCompute to upgrade to a supported version.

9.6 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,as64 MB is the by default size.. The range is 0 < split_size <= (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*.

9.7 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]=∞).

- 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] + 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> {

    @Override
    public void combine(LongWritable vertexId, LongWritable combinedMe
ssage,
        LongWritable messageToCombine) throws IOException {
      if (combinedMessage.get() > messageToCombine.get()) {
        combinedMessage.set(messageToCombine.get());






  public static class SSSPVertexReader extends
      GraphLoader<LongWritable, LongWritable, LongWritable, LongWritab
le> {

    @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] + 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/TotalNumVertices.
- Iteration formula: PageRank(i) = 0.15/TotalNumVertices + 0.85 x sum. Sum indicates the sum of PageRank(j)/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. PageRank(j)/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, DoubleWrit
able> 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, DoubleWrit
able> 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, DoubleWrit
able> 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 edgs 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 `PageRank(i) = 0.15/TotalNumVertices + 0.85 x 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<KmeansAggr
Value> {

    @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.getMaxIter
ation() - 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()) {
          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> [maxIterati
on]");

  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.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 transposeN
eighbors.

 * 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(" inNeighbores: " + 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().setSccID(getId());
        if (this.getValue().getInNeighbors().size() == 0 ||
            this.getNumEdges() == 0) {
          this.getValue().setActive(false);

        break;
```

```
      case STAGE_FW_START:
        this.getValue().setSccID(getId());
        context.sendMessageToNeighbors(this, this.getValue().getSccID
());
        break;
      case STAGE_FW_REST:
        long minSccID = Long.MAX_VALUE;
        for (LongWritable msg : msgs) {
          if (msg.get() < minSccID) {
            minSccID = msg.get();


        if (minSccID < this.getValue().getSccID().get()) {
          this.getValue().setSccID(new LongWritable(minSccID));
          context.sendMessageToNeighbors(this, this.getValue().
getSccID());
          this.getValue().setUpdated(true);
        } else {
          this.getValue().setUpdated(false);

        break;
      case STAGE_BW_START:
        if (this.getId().equals(this.getValue().getSccID())) {
          for (Writable neighbor : this.getValue().getInNeighbors().
getAll()) {
            context.sendMessage((LongWritable)neighbor, this.getValue
().getSccID());

          this.getValue().setActive(false);

        break;
      case STAGE_BW_REST:
        this.getValue().setUpdated(false);
        for (LongWritable msg : msgs) {
          if (msg.equals(this.getValue().getSccID())) {
            for (Writable neighbor : this.getValue().getInNeighbors().
getAll()) {
              context.sendMessage((LongWritable)neighbor, this.
getValue().getSccID());

            this.getValue().setActive(false);
            this.getValue().setUpdated(true);
            break;


        break;

      context.aggregate(0, getValue());

    @Override
    public void cleanup(
        WorkerContext<LongWritable, MyValue, NullWritable, LongWritab
le> context)
        throws IOException {
      context.write(getId(), getValue().getSccID());



   * 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 SCCAggrValue implements Writable {
```

```
    IntWritable stage = new IntWritable(STAGE_TRANSPOSE_1);
    BooleanWritable updated = new BooleanWritable(false);
    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));
```

```
      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.SSSP.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, LongWritab
le> 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:

   * | 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 combinedMe
ssage,
        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
        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. Different from the classification algorithm that processes discrete prediction,

the regression algorithm can predict the continuous value type. The linear regression algorithm defines the loss function as the sum of the least square errors of the sample set. It minimizes the loss function to calculate the weight vector.

A common solution is gradient descent that:

1. Initialize the weight vector to give descent speed rate and iterations (or iteration convergence condition).
2. Calculate the least square error for each sample.
3. Get the sum of the least square error, update the weight based on the descent speed rate.
4. Repeat iterations until convergence.

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 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.lost = 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.lost = new DoubleWritable(gradient.lost.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.lost);
      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_N
um")) == 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 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, VertexChan
ges, 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, LongWritable> 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, LongWritab
le> {

     * 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, LongWritab
le> 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;
      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, VertexChan
ges, 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, LongWritable> 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, LongWritab
le> {

     * 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, LongWritab
le> 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 duplicate, 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 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 Lightning

10.1 Lightning 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 Tablueu 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

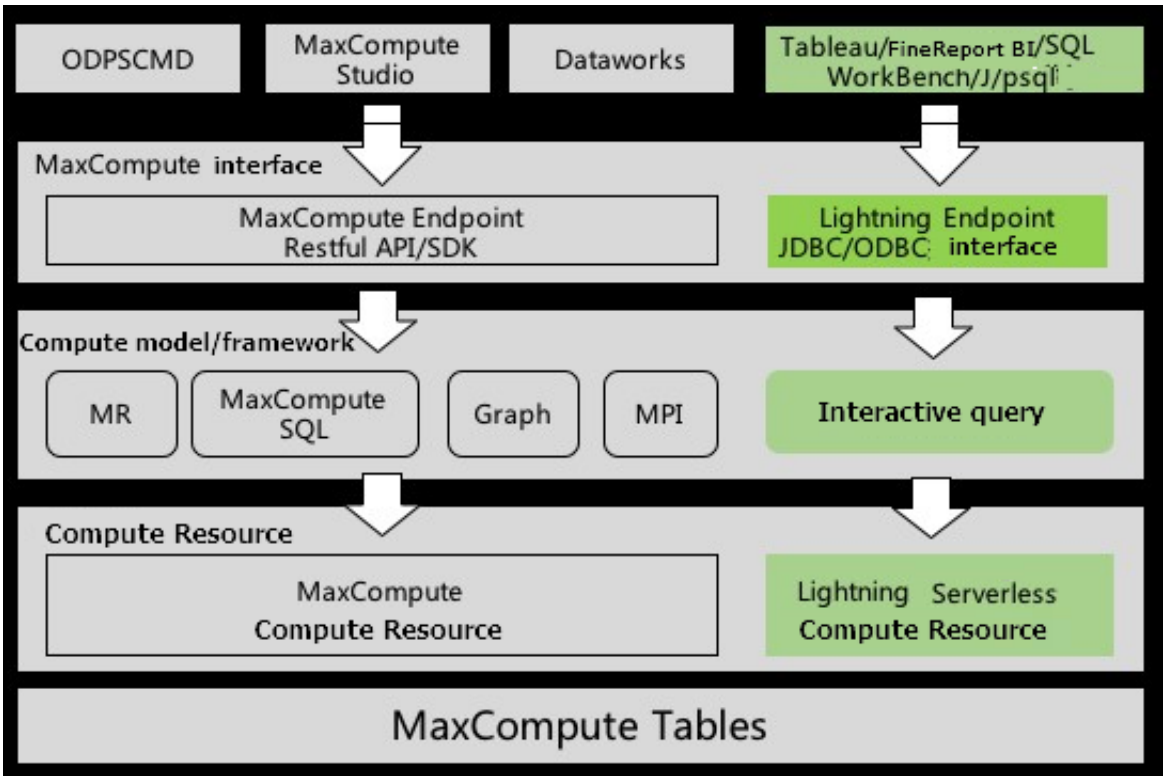
MaxCompute Lightning provides serverless computing services based on existing MaxCompute computing resources. To perform queries, you only need to establish connections to MaxCompute projects using MaxCompute Lightning.

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.

10.2 Activate Lightning service

MaxCompute Lightning provides interactive query services for MaxCompute, using MaxCompute Lightning requires that MaxCompute has been activated and a project has been created.

MaxCompute Lightning service is currently in beta and is not open to users on the entire network.

The MaxCompute Lightning service is activated by default for MaxCompute projects after the beta ends.

10.3 Quick Start

10.3.1 Guide description

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.

10.3.2 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.

10.3.3 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.

10.3.4 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 **Connection** > **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 `lightning.cn-shanghai.maxcompute.aliyun.com` as the endpoint for the China East 2 region.
Port	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*.

10.4 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.

Table 10-1: Service connection status in different regions with external network

Region	Service status	External network endpoint
China East 1	Beta	lightning.cn-hangzhou.maxcompute.aliyun.com
China East 2	Beta	lightning.cn-shanghai.maxcompute.aliyun.com
China North 2	Beta	lightning.cn-beijing.maxcompute.aliyun.com
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	-

Table 10-2: Service connection status in different regions with classic network

Region	Service status	Classic network endpoint
China East 1	Beta	lightning.cn-hangzhou.maxcompute.aliyun-inc.com
China East 2	Beta	lightning.cn-shanghai.maxcompute.aliyun-inc.com
China North 2	Beta	lightning.cn-beijing.maxcompute.aliyun-inc.com
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	-

Table 10-3: Service connection status in different regions with VPC network

Region	Service status	VPC endpoint
China East 1	Beta	lightning.cn-hangzhou.maxcompute.aliyun-inc.com
China East 2	Beta	lightning.cn-shanghai.maxcompute.aliyun-inc.com
China East 2 Financial Cloud		lightning.cn-shanghai-finance.maxcompute.aliyun-inc.com

Region	Service status	VPC endpoint
China North 2	Beta	lightning.cn-beijing.maxcompute.aliyun-inc.com
Asia Pacific SE 1	Beta	lightning.ap-southeast-1.maxcompute.aliyun-inc.com
Other regions	Not activated	-

10.5 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.

10.5.1 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
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 `lightning.cn-shanghai.maxcompute.aliyun.com`
port	443	-
database	Name of a MaxCompute project	-
User	Access Key ID of the user	-
password	Access Key Secret of the user	-

Parameter	Value	Description
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 `prepareThreshold=0`.

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.

10.5.2 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.

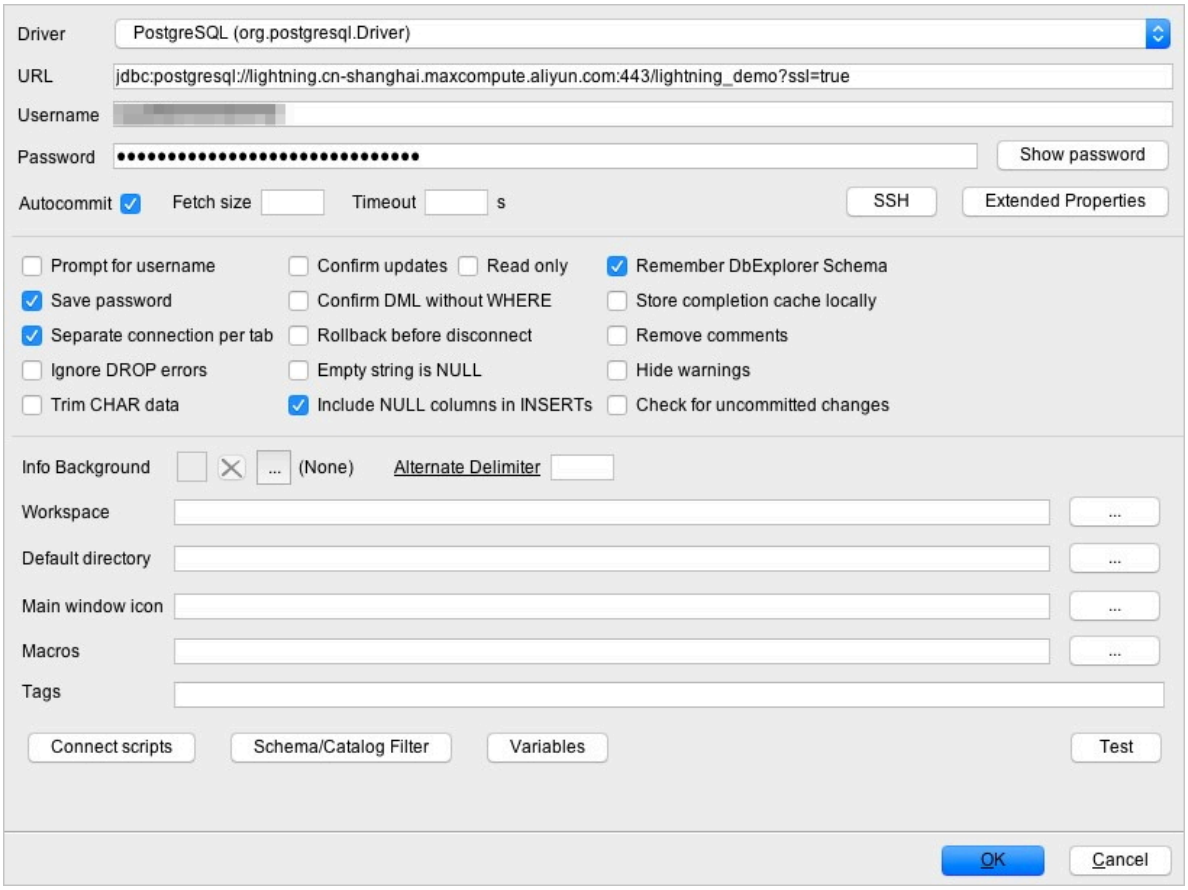
Parameter	Description
Database address	Enter the endpoint for the region of MaxCompute Lightning. You can enter the endpoint for a public network, classic network, or VPC network.
Database	Enter the name of the to-be-accessed MaxCompute project.
Schema	MaxCompute project name
User name/Password	User Access Key ID/Access Key Secret.

SQL Workbench/J

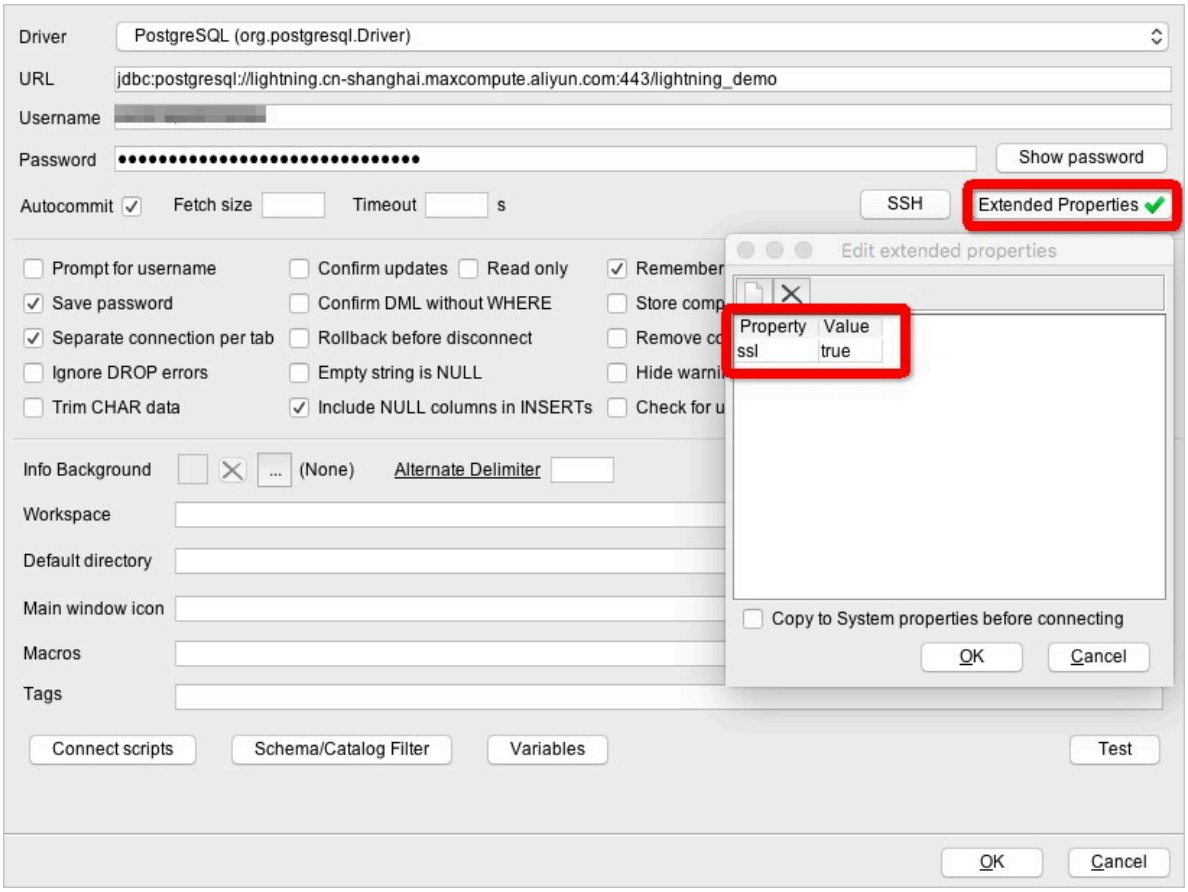
SQL Workbench/J is a widely used free and cross-platform SQL query tool. This tool can be connected to MaxCompute Lightning using the PostgreSQL driver.

1. *Download* and install SQL Workbench/J.
2. Start SQL Workbench/J, establish a database connection.

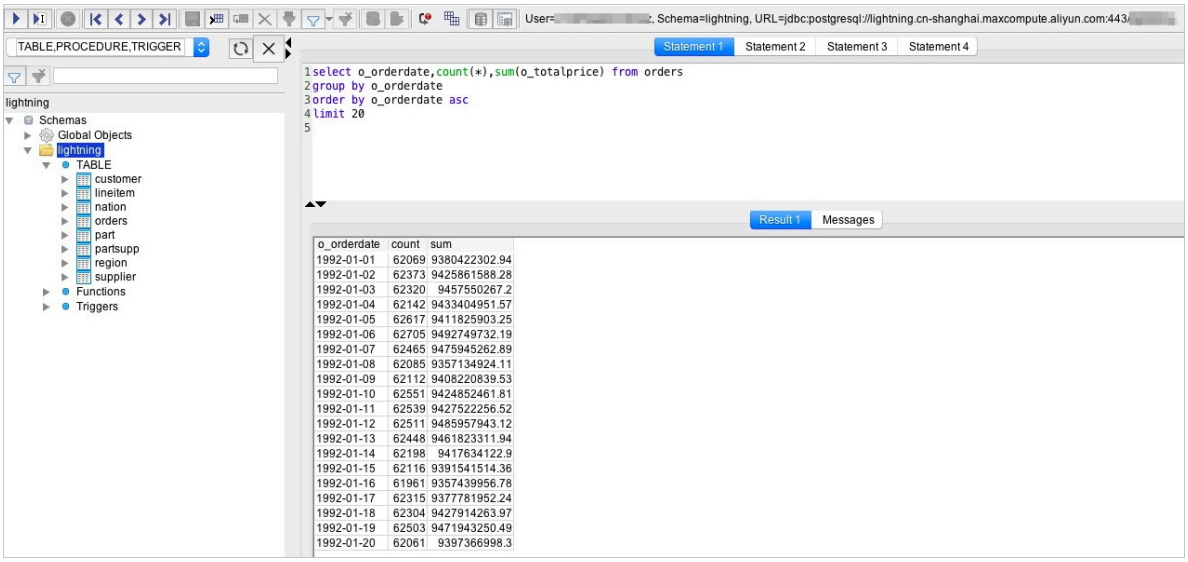
   Select the PostgreSQL driver, connect SQL Workbench/J to the MaxCompute Lightning URL of a project. You must enter the Access Key ID and Access Key Secret of the user.



   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.

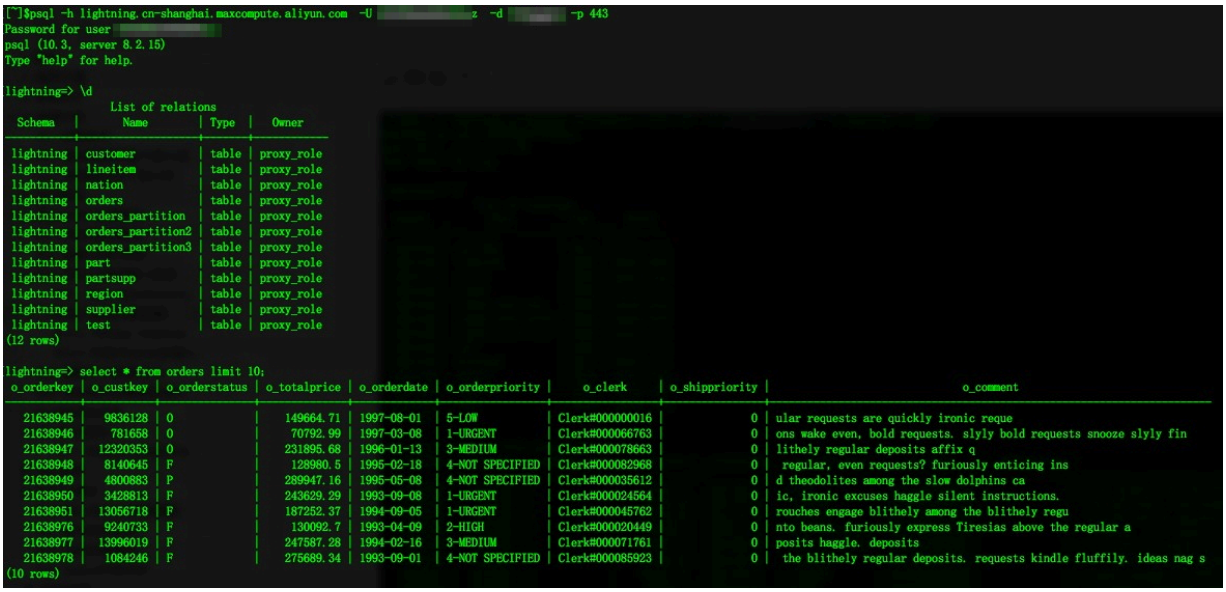
```
psql -h <endpoint> -U <userid> -d <databasename> -p <port>
```

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_INTO' value='no' />
<customization name='CAP_SELECT_TOP_INTO' 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	`jdbc:postgresql://<MaxCompute Lightning Endpoint>:443/<Project_Name>? ssl=true&prepareThreshold=0` For example, `jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/lightning_demo? ssl=true&prepareThreshold=0`
User name/Password	User Access Key ID and Access Key Secret

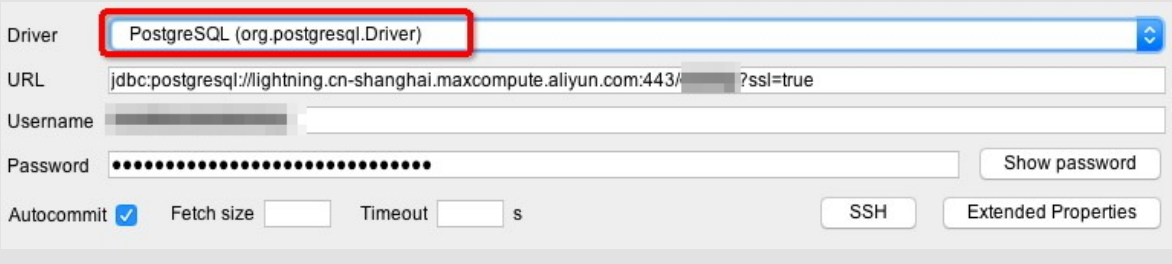
10.5.3 JDBC driver

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.

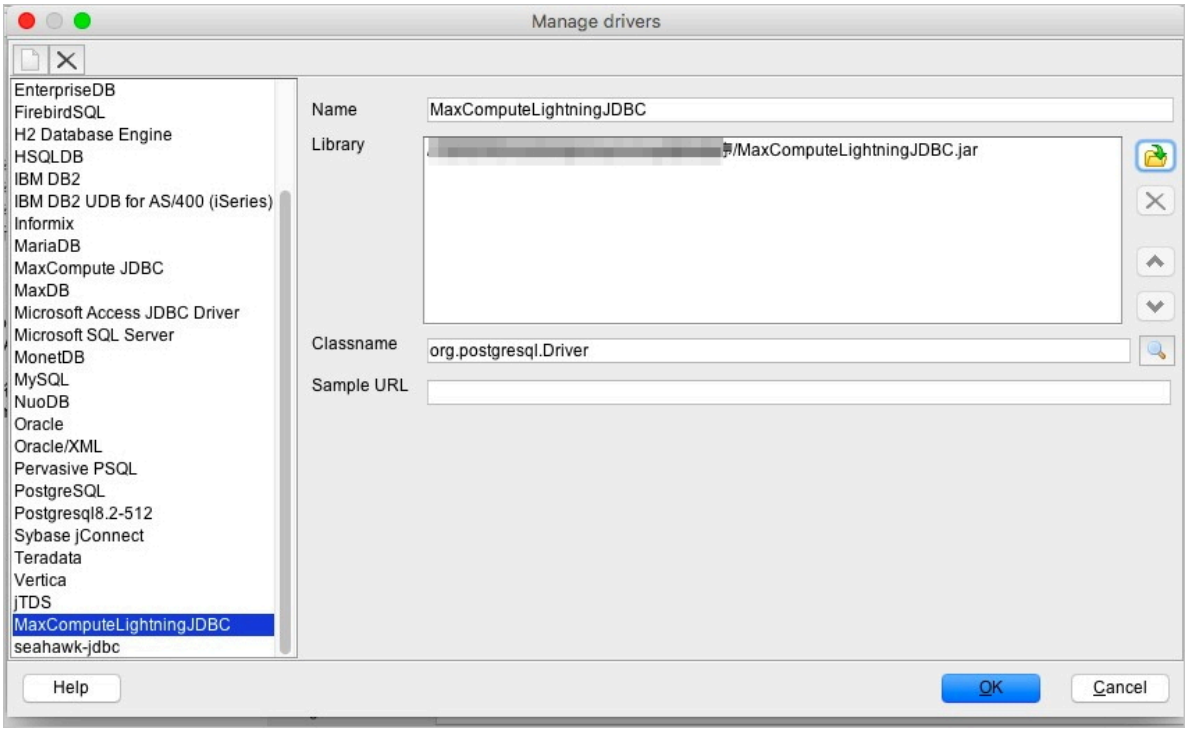
1. *JDBC* drivers from the PostgreSQL official website.

   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.

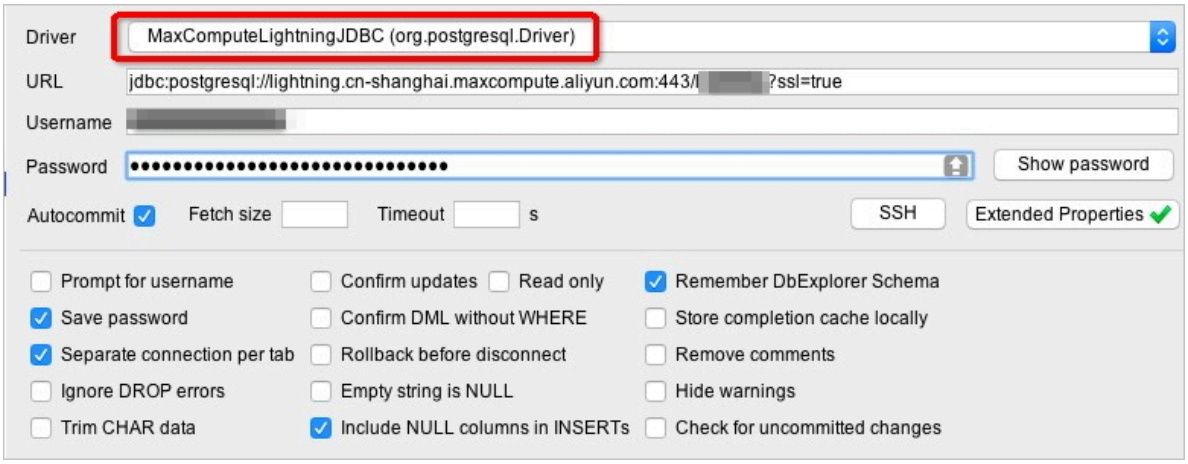
   

2. *JDBC drivers* optimized by Alibaba Cloud MaxCompute Lightning

   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.



10.6 SQL reference

Based on the official PostgreSQL function, MaxCompute Lightning adds the following built-in functions.

Query syntax

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, see *PostgreSQL documentation*.

Function

The MaxCompute Lightning query engine is based on PostgreSQL 8.2 supports builtin funtion, for more information, see *PostgreSQL documentation*.

Based on the official PostgreSQL function, MaxCompute Lightning adds the following builtin functions.

- **MAX_PT**

  Command format

  ```
  max_pt(table_full_name)
  ```

  Command description

  For partitioned tables, this function returns the maximum value of the level-one partition of the partitioned table, sorted alphabetically, and there is a corresponding data file under the partition .

  Parameter description

  table_full_name: String type, used to specify the table name (must carry the project name, such as prj.src), you must have read access to this table.

  Return value

  Returns the value of the largest level-one partition.

  Example

  Suppose tbl is a partition table, the corresponding partition is as follows, and both contain data files:

  ```
  pt =‘20120901’
  pt =‘20120902’
  ```

  Then the partition max_pt returns the value of '20120902' in the following statement, and the MaxCompute SQL statement reads the data under the pt=‘20120902’ partition.

  ```
  select * from tbl where pt=max_pt('myproject.tbl');
  ```

10.7 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.

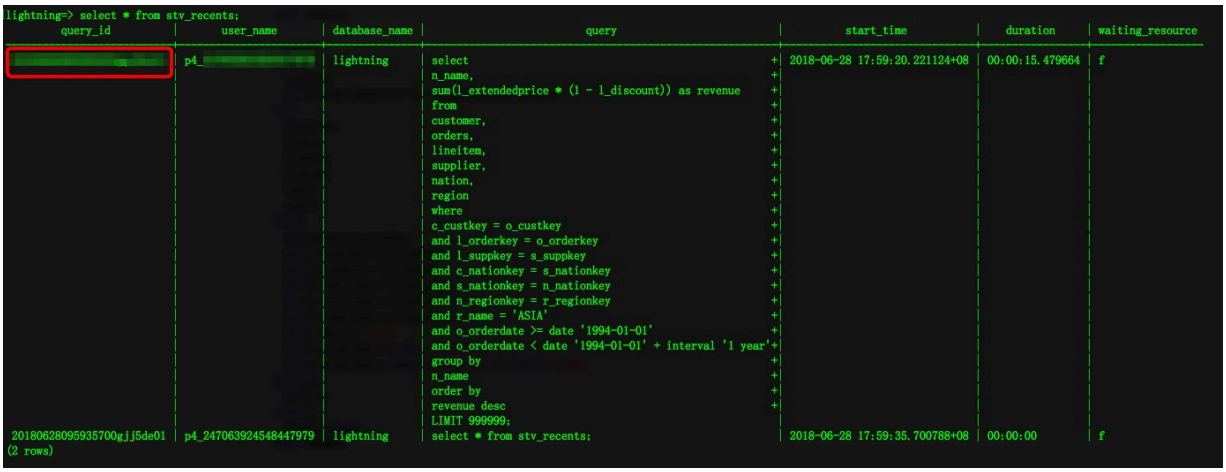
View running queries

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.

10.8 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.

10.9 Lightning FAQs

This article will help you organize common problems in the application of MaxCompute Lightning.

- Q: How can I query data using MaxCompute Lightning when I have not created any tables?

  A: You need to use the DataWorks or odpscmd client tool to create tables for a MaxCompute project and then upload the data. You can access the project using MaxCompute Lightning and query the tables in the project.

- Q: What are the limits on the amount of data that I can query? What is the limit to the amount of queried data MaxCompute Lightning can process and still show excellent performance?

  A: Currently, a maximum of 1 TB data can be scanned for a table in each query. Of course, less of queried data will provide better query performance.

  > **Note:**
  > We recommend that the table data to be scanned does not exceed 100 GB. Query performance gradually decreases with the increase of data volume. If the queried data exceeds 100 GB, MaxCompute SQL is recommended for better performance.

- Q: What should I do if I receive the following error message when using BI tools to drag a partitioned table for analysis: `ERROR: AXF Exception: specified partitions count in odps table: <project_name.table_name> is: xxx, exceeds the limitation of xxx, please add stricter partition filter`.

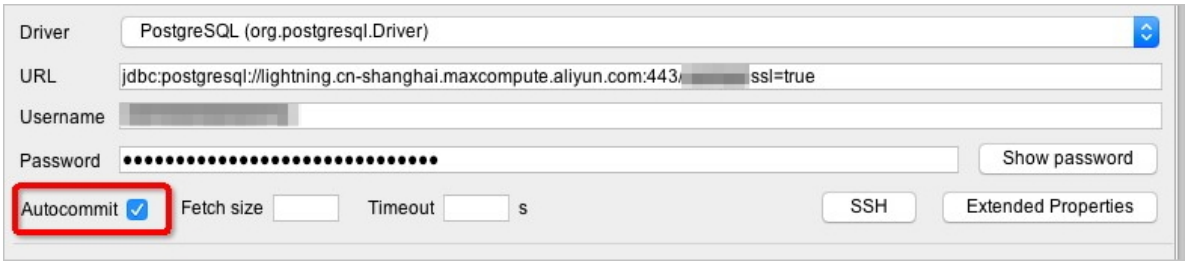
  A: MaxCompute Lightning limits the number of partitions for a partitioned table to ensure the query performs efficiently. A maximum of 1,024 partitions can be scanned for a table in each query. With some BI tools, you can select tables for analysis using the drag-and-drop method . In this way, you are not able to specify partition settings before the analysis. This may cause the number of partitions to be scanned to exceed the limit, triggering the report of an error from MaxCompute Lightning. We recommend that you process the to-be-queried tables before the analysis. You can either convert partitioned tables into non-partitioned tables or reduce the number of partitions to a value lower than 1,024.

- Q: Why is `ERROR: SSL required` displayed during the connection to MaxCompute Lightning?

  A: MaxCompute Lightning requires SSL connections and therefore users must use SSL connections. If you use a client tool, you can select the SSL connection check box. If SSL connections cannot be selected in the client tool, you can add the SSL parameter to the JDBC URL. In the JDBC URL, you must enter the endpoint of the region where your project belongs and the name of the connected project, for example, `jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/myproject? ssl=true`.

- Q: What should I do when I receive the following error message during a query using the SQL Workbench/J client: `Error:current transaction is aborted,commands ignored until end of transaction block`.

  A: Select the **Autocommit** check box in the client.


Driver
PostgreSQL (org.postgresql.Driver)
URL
jdbc:postgresql://lightning.cn-shanghai.maxcompute.aliyun.com:443/ ssl=true
Username
Password
Show password
Autocommit
Fetch size
Timeout
s
SSH
Extended Properties

11 MaxCompute Manager

When you start MaxCompute pre-payment, you will encounter one common problem: you have purchased 150 CUs, however, many of your tasks in pre-paid projects may still have to queue up for a long time. Administrators or operations want to know which tasks have occupied resources, so as to control their tasks properly, such as adjusting the scheduling time according to the corresponding business priority of tasks.

MaxCompute Manager provides pre-payment computing resource monitoring and management. Currently, MaxCompute Manager mainly provides three functions: system status monitoring, resource group allocation, and task monitoring. See the DataWorks document *MaxCompute Manager* for detailed instructions.

Note:

MaxCompute Manager prerequisite:

- You should already have purchased MaxCompute pre-paid CU resources and a quantity of 60 CUs or more. You can only take complete advantage of computing resources and MaxCompute Manager when you have sufficient CUs.