Alibaba Cloud MaxCompute

User Guide

Issue: 20180807

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 intellectual 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.	 **Note:** Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, 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

1 Data upload and download

1.1 Data upload and download

This document describes the process of uploading and downloading MaxCompute system data, including service connection, SDKs, tools, and cloud data migration.

You can use the real-time data tunnel of DataHub or the batch data tunnel of Tunnel to access the MaxCompute system. Both DataHub and Tunnel provide their own SDKs. The SDKs and derivative data upload and data download tools can meet your requirements for data upload and data download in a variety of scenarios.

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

Underlying data tunnels used by these tools include the following:

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

The wide range of data upload and data download tools are applicable for most scenarios of cloud data migration. The following documents introduce the tools, Hadoop data migration, database data synchronization, log collection, and other cloud migration scenarios, which you can reference when you are selecting technical solutions.

1.2 Cloud migration of data

Data upload and data download tools of the MaxCompute platform are applicable for a wide range of cloud data migration scenarios. This document introduces some typical scenarios.

Hadoop data migration

You can use Sqoop or DataWorks for Hadoop data migration.

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

Database synchronization

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

- For offline batch data synchronization, you can use DataWorks which 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. For a sample scenario, see *flume collects web site log data to maxcompute* and *sea volume log data analysis and application*.

1.3 Tools

The MaxCompute platform supports a wide range of tools. The source code for most tools can be found on GitHub, the open-source community for data uploading and data downloading. Different tools are applicable to different scenarios, and the tools are divided into two types. The two types are Alibaba Cloud DTplus products and open-source products. This document provides a brief description of 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, ApsaraDB for RDS (MySQL, SQL Server, and PostgreSQL), Oracle, FTP, ADS (AnalyticDB), OSS, Memcache, and DRDS. For more information, see *Data synchronization introduction*, and for usage methods, see *Create a data synchronization task*.

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

Note:

This is an open-source *GitHub project*.

- **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. For more information, see *Create a job to synchronize data from ApsaraDB for RDS to MaxCompute*.

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. For more information, see *MaxCompute Sqoop*.

Note:

This is an open-source *GitHub project*.

- **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 *GitHub project*.

- **Flume**

  Apache Flume is a distributed and reliable system which can efficiently collect, aggregate, and move 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 MaxCompute tables. For more information, see *flume_plugin*.

Note:

This is an open-source *GitHub project*.

- **Fluentd**

  Fluentd is an open-source software product used to collect logs, including application logs, system logs, and access logs, from various sources. It allows you to select plugins 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/StreamCompute, it allows you to easily create an all-in-one streaming data solution 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 Oracle database data to DataHub in real time and archive the data in MaxCompute tables.

Note:

This is an open-source *GitHub project*.

1.5 Import or export data using the Data Integration function

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

Prerequisites

Before importing or exporting data, make sure you have completed the preparatory operations. 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** in the actions column of the relevant project in the **Project List**.
2. Click **Data Integration** in the top navigation bar to go to the **Data Source** page.
3. Click **Add Data Source**.
4. Enter relevant configurations in the data dialog box.

   Configurations:

   - **`Data source name`**: Contains letters, numbers, and underlines. It must begin with a letter or an underline, 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 MySQL.
   - **`ODPS Endpoint`**: Read-only by default. The setting is automatically read from the system configuration.
   - **`MaxCompute project name`**: Identifies 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. Click **Test Connectivity**.
6. If the connectivity test is successful, click **Save**.

Note:
For the configuration of other data sources, see *Data source configuration*.

Use Data Integration to Import Data

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 the target.

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

   Configurations:

   - **Partition information**: You must 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. You must 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 tunnel.

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

   Configurations:

   - **`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 do not set a large value for the concurrent job count to avoid interfering 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 save.

   After configuration, you can scroll up or down to view the task configurations. If no errors found, 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. You can also click **Submit** to submit the synchronization task to the scheduling system of Data IDE. 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

You can 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"
```

Reference Documentation

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

1.6 Tunnel SDK

1.6.1 Summary

MaxCompute Tunnel is the data tunnel of MaxCompute. You can use Tunnel to upload data to or download data from MaxCompute. Tunnel only supports table data uploading and data downloading.

MaxCompute provides *Data upload and download tools* programmed based on the Tunnel SDK.

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:

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

This document describes the main interfaces of Tunnel SDK, which may vary with the SDK version.

Main 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. When you download data with MaxCompute Tunnel using intranet of Alibaba Cloud, no traffic fee is incurred. The intranet address is only valid for cloud products in the Hangzhou region.
TableTunnel.UploadSession	Indicates a session used to upload data to a MaxCompute table.

Main interface	Description
TableTunnel.DownloadSession	Indicates a session used to download 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*.

1.6.2 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.

This 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);
```

Description:

- Life cycle: Starts from TableTunnel instance creation and ends with the completion of the program.
- Provides the method for creating uploading and downloading objects.
- The process of uploading and downloading a table or partition is called as a session. A session consists of one or more HTTP requests to the Tunnel RESTful API.

- The uploading session of TableTunnel is INSERT INTO semantics, which means that sessions that upload the same table or partition do not affect each other. The upload of each session is located in different directories.
- In an uploading session, each RecordWriter corresponds to an HTTP Request, identified by a block id, corresponding to a file on the service side (The block id is the corresponding file name).
- In a session, opening RecordWriter multiple times with the same block id results in overwritin g. The data uploaded by the last RecordWriter calling close() is retained. This feature can be used for retransmissions when block uploading fails.

The TableTunnel Interface implementation process is as follows:

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 directory where the corresponding table resides, and updates the table meta. This means that data moving into the table is visible to other MaxCompute tasks (such as SQL, MR).

The TableTunnel Interface restrictions are as follows:

- 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. If massive data results in the transmission time exceeding 24 hours, you must split them into multiple sessions.
- The HTTP Request timeout for RecordWriter is 120 seconds. If there is no data flow through the HTTP connection within 120 seconds, the service automatically closes the connection.

  Note:

  It should be noted that there is an 8 KB buffer for HTTP itself, so we cannot make sure that there is data flow through an HTTP connection when you call RecordWriter.write() each time. TunnelRecordWriter.flush() can forcibly flush data from the buffer.
- For the scenario that logs are written into MaxCompute, the RecordWriter can easily time out because the data arrives unpredictably. At this time:

- We do not recommend that you open a RecordWriter for each piece of data(Because each RecordWriter corresponds to a file, too many small files can seriously affect the MaxCompute performance).
- We recommend that you can call a RecordWriter to write data in a batch when your code cache data size exceeds 64 MB.

- RecordReader timeout is 300 seconds.

1.6.3 UploadSession

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:

- **Life cycle**: Starts from the creation of the Upload instance and ends with the completion of the upload process.
- **Create Upload instance**: You can create an instance by **Calling the Constructor** or by using **TableTunnel**.
  - Request method: Synchronous.
  - The server creates a session for this upload instance and generates a unique UploadId for the upload. Obtain this ID using the **`getId`** on the client.
- **Upload data**:
  - Request method: Synchronous.
  - Call the **`openRecordWriter`** method to generate a RecordWriter instance. The blockId identifies the data to be uploaded and describes its location in the table within the value

range of [0,20000]. When data fails to be uploaded, you can reupload it based on the blockId.

- **View upload**:
  - Request method: Synchronous.
  - Call **`getStatus`** to obtain the current upload status.
  - Call **`getBlockList`** to obtain the successfully uploaded blockId list. You can compare this with the upload blockId list to find and reupload failed blockIds.
- **End upload**:
  - Request method: 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, you can try again.
- Seven kinds of status are described as follows:
  - UNKNOWN: The initial value when the server creates a session.
  - NORMAL: The upload object is created successfully.
  - CLOSING: When the **`complete`** method (end upload) is called, the server changes the status to CLOSING.
  - CLOSED: After completing upload (which means to move data to the directory where result table is).
  - EXPIRED: The upload has timed out.
  - CRITICAL: Service error.

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, you cannot 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 life cycle 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, the Writer becomes unavailable and you must open a new one.
- 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.

1.6.4 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)
```

Download objects:

- **Life cycle**: Starts from the creation of the Download instance and ends with the completion of data download.
- **Create Download instance**: You can create an instance by Calling the Constructor or by using TableTunnel.
  - Request method: Synchronous.
  - The server creates a session for this download instance and generates a unique DownloadId for the download. Obtain this ID using the **`getId`** on the client.
  - This operation results in high costs. The server creates an index for the data files. In case of a large amount of files, this may take a long time.
  - At the same time, the server returns the total number of Records and starts multiple concurrent downloads based on this value.
- **Download data**:
  - Request method: 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 method: Synchronous.
  - Call **getStatus** to obtain the current download status.
- The four types of status are as follows:
  - UNKNOWN: The initial value when the server creates a session.
  - NORMAL: Create Download object succeeds.
  - CLOSED: After downloading.
  - EXPIRED: The download has timed out.

1.6.5 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. If a block fails to upload, you must re-upload all blocks.
4. After uploading all blocks, provide the uploaded blockid list to the server for verification. This is done by calling **session.commit([1,2,3,…])**.

   The connection time-out and other restrictions on server block management complicate the upload process logic. Therefore, the SDK provides a more advanced RecordWriter—TunnelBufferWriter interface to simplify the process.

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;
```

TunnelBufferedWriter objects:

- Life cycle: Starts from RecordWriter creation and ends with the completion of data upload.

- Create TunnelBufferedWriter instance: Call **openBufferedWriter** interface of UploadSession to create an instance.
- Data upload: Call the Write interface. Data is first written to the local cache. After 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: You can use the setBufferSize interface to modify the size of memory (bytes), occupied by the buffer preferably greater than 64 MB to prevent the server from generating too many small files, which may affect 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);
```

1.7 Bulk data channel SDK example

1.7.1 Example

- MaxCompute provides two service addresses for you to choose from. The Tunnel service address you select 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 reference. Note the differences for different versions.

1.7.2 Example for uploading

```
import java.io.IOException;
 import java.util.Date;
 import com.aliyun.odps.Column;
 import com.aliyun.odps.Odps;
```

```
 import com.aliyun.odps.PartitionSpec;
 import com.aliyun.odps.TableSchema;
 import com.aliyun.odps.account.Account;
 import com.aliyun.odps.account.AliyunAccount;
 import com.aliyun.odps.data.Record;
 import com.aliyun.odps.data.RecordWriter;
 import com.aliyun.odps.tunnel.TableTunnel;
 import com.aliyun.odps.tunnel.TunnelException;
 import com.aliyun.odps.tunnel.TableTunnel.UploadSession;
 public class UploadSample {
         private static String accessId = "<your access id>";
         private static String accessKey = "<your access Key>";
         private static String odpsUrl = "http://service.odps.aliyun.
com/api";
         private static String tunnelUrl = "http://dt.cn-shanghai.
maxcompute.aliyun-inc.com";
                         //The tunnelURL must be set if you need to
connect internal network, otherwise, the system uses public network as
 default. The example shows the Tunnel Endpoint of classical network
in 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>";
         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());
                         TableSchema schema = uploadSession.getSchema
();
                          // After preparing data, open a Writer to
start writing data. The prepared data is written to one block.
                          // When the data written to individual
 blocks is too small, the system will produce a large number of
small files, seriously degrading computing performance. We strongly
 recommend over 64 MB of data be written each time (up to 100 GB of
data can be written to the same block).
                          // You can use the average data volume and
record count to estimate the total value. For example: 64MB < Average
data size x Record count < 100GB.
                         RecordWriter recordWriter = uploadSession.
openRecordWriter(0);
                         Record record = uploadSession.newRecord();
                         for (int i = 0; i < schema.getColumns().size
(); i++) {
                                 Column column = schema.getColumn(i);
                                 switch (column.getType()) {
                                 case BIGINT:
                                         record.setBigint(i, 1L);
```

```
                                        break;
                                    Case Boolean:
                                        record.setBoolean(i, true);
                                        break;
                                    case DATETIME:
                                        record.setDatetime(i, new
Date());
                                        break;
                                    case DOUBLE:
                                        record.setDouble(i, 0.0);
                                        break;
                                    case STRING:
                                        record.setString(i, "sample
");
                                        break;
                                    default:
                                        throw new RuntimeException("
Unknown column type: "
                                                                + column.
getType());
                                    }
                            }
                            for (int i = 0; i < 10; i++) {
                                     // Writes data to the server. Each 8
 KB of data written triggers a network transmission.
                                    // If no network transmission occurs
 for 120 seconds, the server closes the connection. At this time, the
Writer becomes unavailable and you must write data again.
                                    recordWriter.write(record);
                            }
                            recordWriter.close();
                            uploadSession.commit(new Long[]{0L});
                            System.out.println("upload success!") ;
                    } catch (TunnelException e) {
                            e.printStackTrace();
                    } catch (IOException e) {
                            e.printStackTrace();
                    }
            }
 }
```

Constructor:

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

Parameter descriptions:

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

In this program, the configuration must be as follows:

```
private static String partition = “pt=’XXX’,ds=’XXX’”;
```

1.7.3 Example for downloading

```
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";
                         //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>";
         public static void main(String args[]) {
                 Account account = new AliyunAccount(accessId,
accessKey);
                 Odps odps = new Odps(account);
                 odps.setEndpoint(odpsUrl);
                 odps.setDefaultProject(project);
                 TableTunnel tunnel = new TableTunnel(odps);
                 tunnel.setEndpoint(tunnelUrl);//set tunnelUrl
                 PartitionSpec partitionSpec = new PartitionSpec(
partition);
                 try {
                         DownloadSession downloadSession = tunnel.
createDownloadSession(project, table,
                                         partitionSpec);
                         System.out.println("Session Status is : "
                                         + downloadSession.getStatus
().toString());
                         long count = downloadSession.getRecordCount
();
                         System.out.println("RecordCount is: " + count
);
                         RecordReader recordReader = downloadSession.
openRecordReader(0,
                                         count);
                         Record record;
                         while ((record = recordReader.read()) ! =
null) {
                                 consumeRecord(record, downloadSession
.getSchema());

                         recordReader.close();
                 } catch (TunnelException e) {
                         e.printStackTrace();
                 } catch (IOException e1) {
                         e1.printStackTrace();


         private static void consumeRecord(Record record, TableSchema
schema) {
```

```
                for (int i = 0; i < schema.getColumns().size(); i++)
 {
                        Column column = schema.getColumn(i);
                        String colValue = null;
                        switch (column.getType()) {
                        case BIGINT: {
                                Long v = record.getBigint(i);
                                colValue = v == null ? null : v.
toString();
                                break;

                        case BOOLEAN: {
                                Boolean v = record.getBoolean(i);
                                colValue = v == null ? null : v.
toString();
                                break;

                        case DATETIME: {
                                Date v = record.getDatetime(i);
                                colValue = v == null ? null : v.
toString();
                                break;

                        case DOUBLE: {
                                Double v = record.getDouble(i);
                                colValue = v == null ? null : v.
toString();
                                break;

                        case STRING: {
                                String v = record.getString(i);
                                colValue = v == null ? null : v.
toString();
                                break;

                        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();
```

In this example, data is directly printed using System.out.println to facilitate testing. In actual use, you can directly output the data to a text file.

1.7.7 Example for BufferedWriter uploading

```
// 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();
```

1.9 Connection to data tunnel service

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. You must select the proper address or endpoint for your network or you are unable to send requests to the service. At the same time, different network connections also have an impact on your *billing*.

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

2 Common commands

2.1 Overview of Common commands

This module will show you in detail 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 via the console commands and SDK.

Note:

- The Common commands introduced in this module are mainly targeted at the new version of the console.
- If you want to learn how to install and configure clients, see Quick Start.
- For more information about the SDK, see maxcompute. SDK introduction.

2.2 Project Operations

Enter the project

Command format:

```
use <project_name>;
```

Action:

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

Example:

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

Note:

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

After running the command, a user can access the objects of this project. 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 project, such as ‘my_project2’, through the project ‘my_project’, you must first specify the project 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 projects. You can use the MaxCompute console for additional configurations and operations as needed.

2.3 Table Operations

This article shows how to use the common commands to operate tables in the MaxCompute client.

If you want to operate a table, you can use common commands in the client, and you can also easily collect tables, apply permissions, and view partitions through the visible data table management 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:

- The table name and column name are both case insensitive.

- A table name and column name obey the same naming conventions as follows: The name can be up to 128 bytes in length and can contain letters, numbers, and underscores '_'.
- The comment content is the effective string, and it can be up to 1,024 bytes in length.
- [LIFECYCLE days] The parameter 'days' refers to the lifecycle time and must be a positive integer. Unit is 'day'.
- Suppose that the table 'table_name' is 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).
- Suppose that the table 'table_name' is a partition table. MaxCompute judges whether to recycle the table according to LastDataModifiedTime of each partition. Unlike for non-partitioned tables, a partitioned table is not dropped after the last partition is recycled. The ' lifecycle' can only be set at the table level, but not at the partition level.

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

Statement 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 the information of specified table, including:

- Owner: The owner of the table.
- Project: The project that table belongs to.
- CreateTime: The creation time of the table.
- LastDDLTime: The last DDL operation.
- LastModifiedTime: The last time of table modification.
- InternalTable: It indicates the object to be described is table and always shows YES.
- 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 always YES. Similarly, the option 'Size' can be replaced by ViewText. For example: `select * from src`. For more information about view, see Create View.

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 '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.
- Alibaba Cloud is system prompt, indicating the Alibaba Cloud user.
- Odps_user@aliyun.com is the creator of the table 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 Instance

Show instances/Show P

The command format is as follows:

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

```
SHOW INSTANCES [-all];
SHOW P [-all];
```

Action:

The information of instances created by current users is displayed.

Parameters information is as follows:

- **`Startdate`**, **`enddate`**: returns the information about the instances during specified period (from startdate to enddate). The following format must be met: yyyy-mm-dd, precision to the day. Optional parameter, if not specified, returns the information of instances you submitted within three days.
- **`Number`**: Specify the number of instance to be showed.In accordance with the time scheduling, return N (number) instances nearest to the current time. If it is not specified, all instances that meet the requirements are shown. In chronological order, the specified number of instances most recently preceding the current time are returned. The information of all instances meeting requirements is returned.
- **`-all`**:Returns all instances performed under the current project. Note: The user executing the command needs to have list permission for the project.
- The output items: Include StartTime (the time accurate to seconds), RunTime (s), Status (including Waiting, Success, Failed, Running, Cancelled, and Suspended).

InstanceID and corresponding SQL are as following:

```
StartTime RunTime Status InstanceID Query
2015-04-28 13:57:55 1s Success 20150428055754916grvd5vj4 select * from
 tab_pack_priv limit 20;
```

Six kinds of instance status are possible:

- Running
- Success
- Waiting
- Failed, but the data of the target table have not been modified.
- Suspended
- Canceled

Note:

The commands from the preceding example run in MaxCompute client.

Instance status

The command format is as follows:

```
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 current user, exception is returned.

Example:

```
odps@ $project_name>status 20131225123302267gk3u6k4y2;
Success
```

Query the status of an instance which ID is 20131225123302267gk3u6k4y2, and the result is Success.

Note:

The commands from the preceding example run in MaxCompute client.

Top instance

The command format is as follows:

```
TOP INSTANCE;
```

Action:

The job information that is running in the current project is displayed, including ISNTANCEID, Owner, Type, StartTime, Progress, Status, Priority, RuntimeUsage (CPU/MEM), TotalUsage (CPU/MEM), QueueingInfo (POS/LEN) and so on.

Example:

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

Note:

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

Delete an instance

The command format is as follows:

```
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. Instance status must be Running.

Example:

```
odps@ $project_name>kill 20131225123302267gk3u6k4y2;
```

Stop the instance which ID is 20131225123302267gk3u6k4y2.

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 result. To check whether the instance is deleted, use the **`status`** command.

Describe an instance

The command format is as folllows:

```
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@ $project_name> desc instance 20150715103441522gond1qa2;
ID 20150715103441522gond1qa2
Owner ALIYUN$maojing.mj@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 which ID is 20150715103441522gond1qa2.

Note:
The commands from the preceding example run in MaxCompute client.

Wait instance

The command format is as follows:

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

Action:

Get running task information, including logs according to instance ID, and logview link. You can view task details by accessing logview link.

Example:

```
wait 20170925161122379g357ldqp;
ID = 20170925161122379g357ldqp
Log view:
http://logview.odps.aliyun.com/logview/?h=http://service.odps.aliyun.
com/api&p=alian&i=20170925161122379g357ldqp&token=WnlzSGMwZG5vMUZxMGFT
Wk5hUElwYm1jb21VPSxPRFBTX09CTzoxMzI5MzgzMDA0NTQwNjUxLDE1MDcxOTE0MDYsey
JTdGF0ZW1lbnQiOlt7IkFjdGlvbiI6WyJvZHBzOlJlYWQiXSwiRWZmZWN0IjoiQWxsb3ci
LCJSZXNvdXJjZSI6WyJhY3M6b2RwczoqOnByb2plY3RzL2FsaWFuL2luc3RhbmNlcy8yMD
E3MDkyNTE2MTEyMjM3OWczNTdsZHFwIl19XSwiVmVyc2lvbiI6IjEifQ==
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)
```

2.6 Function Operations

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

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

Create a function

The command format is as follows:

```
CREATE FUNCTION AS USING ;
```

Parameters information:

- **`function_name`**: UDF name, which is the 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.
- **`resource_list`**: resources list used by UDF.
  - The resource which contains UDF code must be included in the list.
  - If the your code reads the resource file by 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 comma (,). The resource list must be in double quotation marks.

Example:

Suppose that the Java UDF class org.alidata.odps.udf.examples.Lower is in my_lower.jar. Create the UDF my_lower function as follows:

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

Note:

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

Delete a function

The command format is as follows:

```
DROP FUNCTION ;
```

The example is as follows:

```
DROP FUNCTION test_lower;
```

List functions

The command format is as follows:

```
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 Other Operations

ALIAS command

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

The command format is as follows:

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

Action:

Create alias for a resource.

examples:

```
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.

Set

The command format is as follows:

```
set ["<KEY>=<VALUE>"]
```

Actioin:

You can use the set command to set MaxCompute or a user-defined system variables that affects 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 = false/true  --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 of each
reduce worker. Unit is M and default value is 1024M.
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.sql.mapper.split.size=256
    -- 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 = true/false;  --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

The command format is as follows:

```
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.

SetProject

The command format is as follows:

```
setproject ["="];
```

Action:

- You can use **`setproject`** command to set project attributes.
- If the value of < KEY >=< VALUE > is not specified, the current project attribute configuration is displayed.

The detailed description of project attributes is shown as follows:

Attribute name	Cunfigured permissions	Attribute description	Value range
odps. sql. allow. fullscan	ProjectOwner	item whether to allow 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 project.	IP list, separated bycomma.
odps.table.lifecycle	ProjectOwner	Optional: the lifecycle clause is optional when creating a table . If the user does not set the lifecycle, the table is effective permanently. Required: the lifecycle clause is required. Inherit: if the user does not specify the lifecycle the lifecycle is the value of odps.table. lifecycle.value.	optional/mandatory /inherit
odps.table.lifecycle. value	ProjectOwner	Default lifecycle.	1 ~ 37231(default value)
odps.instance.remain. days	ProjectOwner	How long the instance informatio n is retained.	3–30
READ_TABLE _MAX_ROW	ProjectOwner	The number of data entries returned by running the Select statement in the client.	1 ~ 10000

Take odps.security.ip.whitelist as an example:

MaxCompute supports IP whitelist of the project level.

Note:

- With IP whitelist configured, only the IP (console IP or IP of exit where SDK is located) in the whitelist can access this project.
- After setting the IP white list, you need to wait five minutes before it takes effect.

You can type three formats for the IP list in the whitelist:

- 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.

These three formats can appear in the same command and must be separated by commas (,).

For example the command line tool set the IP white list of methods:

```
setproject odps. security. ip. whitelist = 101.132.236.134, 100.116.0.
0/16,101.132 .236.134-101.132.236.144;
```

If there is no IP address in whitelist, it means whitelist function is disabled.

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

SetProject

The command format is as follows:

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

Cost SQL

The command format is as follows:

```
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:
This information cannot be used as an actual charging standard, can be used only for reference.

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 = 20150715113033121gmsbjxl1
Input: 65727592 Bytes
UDF:0
Complexity: 1.0
```

3 SQL

3.1 SQL Summary

MaxCompute SQL is suitable for Massive data (GB, TB, or EB level) must be processed based on offline batch calculation. It takes several seconds or even minutes to schedule after you submit a job, therefore MaxCompute SQL is most suitable for services which need to process tens of thousands of transactions per second.

The syntax of MaxCompute SQL is similar to SQL. It can be considered as a subset of standard SQL. But MaxCompute SQL is not equivalent to a database, which has no database characteristics in many aspects, such as transaction, primary key constraints, index, and so on. The maximum size of SQL in MaxCompute is 2 MB.

Reserved Word

MaxCompute SQL considers the keywords of SQL statement as reserved words. These cannot be used to name tables, columns, or partitions. If reserved words are used for naming purposes, an error occurs. Reserved words are case insensitive. The reserved words in common use are shown as follows, for the 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 to another one.
- Implicit conversions: MaxCompute automatically performs implicit conversions during 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 table. Specifying the partition can bring lot of conveniences to users. For example, improve SQL running efficiency, reduce the cost, etc. For more information about partition, see *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.

Select Transform

Select The transform function obviously simplifies the reference to the script code, supports languages such as Java, Python, Shell, Perl, and so on, and is easy to write, it is suitable for the implementation of adhoc function. For more information, see the *select transform syntax*.

Currently maxcompute's select transform is fully compatible with hive's syntax, functionality, and behavior, includes input/output row Format and Reader/writer. Most of the scripts on hive can be run directly, some scripts only need a slight change to run.

3.4 DDL SQL

3.4.2 Lifecycle of table

Modify lifecycle of table

MaxCompute provides the function to manage data lifecycle so that user can release storage space and simplify data recycling flow.

Statement format:

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

Note:

- The parameter ‘days’ refers to the lifecycle time and must be a positive integer. Unit is ‘day’.
- Suppose that the table ‘table_name’ is 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 table is modified, the LastDataModifiedTime is updated. So MaxCompute judges whether to recycle this table based on the setting of LastDataMo difiedTime and lifecycle.

- Suppose that the table ‘table_name’ is a partition table. MaxCompute judges whether to recycle the table according to LastDataModifiedTime of each partition.
- Different from 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 a 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, the data at the begining of the month, or the data during the Global Shopping Day period,you can disable the lifecycle function for 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;
```

3.4.3 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.
- It is not allowed writing data into a view, such as: using ‘insert into’ or ‘insert overwrite’ to operate view.
- After a view was created, maybe it is not able to be accessed if the referenced table is altered , such as deleting 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;
```

3.4.4 Column/Partition operation

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 and creating partition columns are not supported.
- If the same name partition has already existed and the option [if not exists] is not specified, return exception.
- Currently, the maximum number of partitions supported in a single table in MaxCompute 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 is indicated.

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
...)
```

Note:

You cannot specify order for the new column. By default, the new column is located 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’ must be an existing column.
- There cannot be a column named ‘new_col_name’ in a 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.
- There cannot be a column named 'new_col_name' in a table.
- The maximum comment content is 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, return an error.
- If the specified partition_col_value does not exist, return an error.
- This operation changes the value of 'LastDataModifiedTime' in a table and now MaxCompute considers the data of 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.

3.5 Insert Operation

3.5.2 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. Once exceeding 256 ways of output, syntax error occurs.
- In a multi insert statement:
  - For a partitioned table, a target partition cannot appear for multiple times.
  - For an unpartitioned table, this table cannot appear for 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 for multiple times.

```
create table sale_detail_multi like sale_detail;
    from sale_detail
        insert overwrite table sale_detail_multi partition (sale_date
='2010', region='china' )
            select shop_name, customer_id, total_price where .....
        insert overwrite table sale_detail_multi partition (sale_date
='2011', region='china' )
            select shop_name, customer_id, total_price where .....
    -- Return result successfully. Insert the data of sale_detail
into the 2010 sales records and 2011 sales records in China region.
    from sale_detail
        insert overwrite table sale_detail_multi partition (sale_date
='2010', region='china' )
            select shop_name, customer_id, total_price
```

```
        insert overwrite table sale_detail_multi partition (sale_date
='2010', region='china' )
            select shop_name, customer_id, total_price;
    -- An error is thrown. The same partition appears for multiple
times.
    from sale_detail
        insert overwrite table sale_detail_multi partition (sale_date
='2010', region='china' )
            select shop_name, customer_id, total_price
        insert into table sale_detail_multi partition (sale_date='
2011', region='china' )
            select shop_name, customer_id, total_price;
    -- An error is thrown. Different partitions within a partition
table cannot have both an ‘insert overwrite’ operation and an ‘insert
 into’ operation.
```

3.5.3 DYNAMIC PARTITION

To ‘insert overwrite’ into a partition table, you can 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. 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 the dynamic partition value for the target table. If the target table only has a 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 characters and Chinese, otherwise exception is thrown. The exception is as follows:

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

- If the destination table has multiple-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.

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 preceding mentioned, 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 through the value of 'region'. This is why the partition is called **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

  ```
  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 lowsubpart
ition 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, an example is as follows:

```
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

3.5.4 VALUES

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

Note:

Currently, INSERT OVERWRITE does not support to insert columns. You can 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 , you can 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…’.

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 |
```

Note:

- values only support constants and don't support functions. Like ARRAY complex types, MaxCompute cannot construct corresponding constants currently. You can modify the statement into

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

  which can achieve the same effect.
- To write datetime or timestamp type through values, you must 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');
  ```

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

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 a anonymous values table in one row and zero columns. In this way, when you want to test some functions, such as your UDF, etc., you do not need to manually create DUAL tables.

3.6 Select Operation

3.6.1 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 the following when using a SELECT statement:

- When using SELECT to read data from a table, you can 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;
  ```

  If you want to read only the shop_name column in sale_detail, use the following statement:

  ```
  select shop_name from sale_detail;
  ```

  You can 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, reducing unnecessary SQL I/O, waste of 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 forbidden 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 the 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;
```

If the entire project is required to allow full table scanning, the switch can be turned on or off by itself (true/false), and the command is as follows:

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

- 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 the whole table. 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:

  - The key using group by can be the name of a column in the input table.
  - 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.
  - 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 about aggregate functions, see *Aggregate Functions*.

- order by: Globally sorts all data based on certain columns. To sort records in descending order, you can 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.

3.6.2 SELECT Sequence

The actual logic execution sequence of SELECT statements written in compliance with the preceding SELECT syntax is different from the standard writing sequence. See the following:

```
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;
```

3.6.3 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 part of the ON condition in SEMI JOIN.

NOT IN SUBQUERY is similar to LEFT ANTI JOIN. However, they have a 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 on the contrary.

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;
```

3.6.4 UNION ALL/UNION [DISTINCT]

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 meeting the conditions are returned, and deduplication of duplicated rows is not implemented.
- **`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 implemented.

UNION An example of the UNION ALL operation:

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

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, names, and types of queried columns corresponding to the UNION ALL/UNION operation must be consistent. If the column names are inconsistent, use the column aliases.
- 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. MaxCompute adopts this action in `set odps.sql.type.system.odps2=true;` currently.

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 results are as follows:

```
| a |

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

3.6.6 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 stored 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.

3.6.10 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 can be rewritten only once, and reused subsequently. The WITH clause in the CTE can specify multiple subqueries that can be repeatedly used like variables in the entire statement. Besides being reused, subqueries do not have to be repeatedly nested.

3.7 DDL SQL

3.8 Insert Operation

3.9 SQL restrictions

Some users may fail to notice specific restrictions and find the service has stopped. The restrictions for MaxCompute SQL include the following:

Boundry name	Maximum value/ Restriction	Class	Description
Length of table name	128 bytes	Length limit	Table names and column names cannot contain special characters. They can contain only English letters (a-z, A-Z), numbers, and underscores (_), and must start with a letter.
Annotation length	1,024 bytes	Length limit	The annotation can contain valid strings of up to 1,024 bytes.
Column definitions	1,200	Quantity limit	One table can contain 1,200 column definitions at most.

Boundry name	Maximum value/ Restriction	Class	Description
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.
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.

Boundry name	Maximum value/ Restriction	Class	Description
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.
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 restrictions of MaxCompute SQL cannot be manually modified or configured.

3.10 Builtin Function

3.10.1 Date Functions

MaxCompute SQL provides the necessary functions to operate datetime types.

DATEADD

Function definition:

```
datetime dateadd(datetime date, bigint delta, string datepart)
```

Usage:

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. T his field value follows 'string' and 'datetime' type conversion agreement, that is, '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 keep the day unchangeable, otherwise 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…If no special description exists, related datetime built-in functions all follow this agreement. And if no special instructions, the part of all datetime built-in functions also supports extended date format: year- 'year'; month-'month' or 'mon'; day-'day'; hour-'hour.

Examples:

```
 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 is only used 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

Function definition:

```
bigint datediff(datetime date1, datetime date2, string datepart)
```

Usage:

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, it causes an exception

Return Value:

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.

The example is as follows:

```
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
```

DATEPART

The command format is as follows:

```
bigint datepart(datetime date, string datepart)
```

Usage:

Extract the value of 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, it causes exception.

Bigint type. If any input is NULL, return NULL.

The example is as follows:

```
datepart('2013-06-08 01:10:00', 'yyyy') = 2013
```

```
 datepart('2013-06-08 01:10:00', 'mm') = 6
```

DATETRUNC

Function definition:

```
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, it causes an exception.

Return Value:

Datetime type. If any input is NULL, return NULL.

The example is as follows:

```
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
```

FROM_UNIXTIME

Function definition:

```
datetime from_unixtime(bigint unixtime)
```

Usage:

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.

Examples:

```
from_unixtime(123456789) = 1973-11-30 05:33:09
```

GETDATE

Function definition:

```
datetime getdate()
```

Usage:

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

Function definition:

```
boolean isdate(string date, string format)
```

Usage:

Judge whether a date string can be converted to a datetime value according to corresponding format string. If the conversion is successful, return TRUE, otherwise return FALSE.

Parameter description:

- **`date`**: date value of String format. If the input is 'bigint', or 'double' or 'datetime', it is be converted to 'string' type. If it is another data type, an exception reported.
- **`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

Function definition:

```
datetime lastday(datetime date)
```

Usage:

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

Function definition:

```
datetime to_date(string date, string format)
```

Usage:

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 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 caused. 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'.

Return Value:

Datetime type, the format is yyyy-mm-dd hh: mi: ss. If any input is NULL, return NULL.

Examples:

```
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')
-- Format is not compatible and exception is thrown.
to_date('Alibaba2010-12*3', 'Alibabayyyy-mm*dd')
-- Format is not compatible and exception is thrown.
to_date('2010-24-01', 'yyyy')
-- Format is not compatible and exception is thrown.
```

TO_CHAR

Function definition:

```
string to_char(datetime date, string format)
```

Usage:

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 caused. In 'format', the date format part is replaced with the corresponding data and other characters are output directly.

Return Value:

String type. Any input parameter is NULL, return NULL.

Examples:

```
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

Function definition:

```
bigint unix_timestamp(datetime date)
```

Usage:

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.

WEEKDAY

Function definition:

```
bigint weekday(datetime date)
```

Usage:

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 regarded as the first day of a week. Its returned value is 0. Days are in ascending order starting at 0. If the day is Sunday, then return 6.

WEEKOFYEAR

Function definition:

```
bigint weekofyear(datetime date)
```

Usage:

Return the nth week of a year which the date is included in. Monday is taken as the first day of a week.

Note:

About 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 indicated.

Return Value:

Bigint type. If the input is NULL, return NULL.

Examples:

```
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.
```

New extended date functions

With the upgrade to 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

Function definition:

```
INT year(string date)
```

Usage:

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.

Examples:

```
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

Function definition:

```
INT quarter(datetime/timestamp/string date )
```

Usage:

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.

Examples:

```
quarter('1970-11-12 10:00:00')=4
```

```
quarter('1970-11-12')=4
```

MONTH

Function definition:

```
INT month(string date)
```

Usage:

Returns the month of a date.

Parameter description:

`date`: String-type date value. Other value types return an exception.

Return Value:

Int type.

Examples:

```
month('2014-09-01')=9
month('20140901')=null
```

DAY

Function definition:

```
INT day(string date)
```

Usage:

Returns the day of a date.

Parameter description:

`date`: String-type date value. Other value types return an exception.

Return Value:

Int type.

Examples:

```
day('2014-09-01')=1
```

```
day('20140901')=null
```

DAYOFMONTH

Function definition:

```
INT dayofmonth(date)
```

Usage:

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.

Examples:

```
dayofmonth('2014-09-01')=1
dayofmonth('20140901')=null
```

HOUR

Function definition:

```
INT hour(string date)
```

Usage:

Returns the hour of a date.

Parameter description:

`date`: String-type date value. Other value types return an exception.

Return Value:

Int type.

Examples:

```
hour('2014-09-01 12:00:00')=12
hour('12:00:00')=12
```

```
hour('20140901120000')=null
```

MINUTE

Function definition:

```
INT minute(string date)
```

Usage:

Returns the minute of a date.

Parameter description:

`date`: String-type date value. Other value types return an exception.

Return Value:

Int type.

Examples:

```
minute('2014-09-01 12:30:00')=30
minute('12:30:00')=30
minute('20140901120000')=null
```

SECOND

Function definition:

```
INT second(string date)
```

Usage:

Returns the second of a date.

Parameter description:

`date`: String-type date value. Other value types return an exception.

Return Value:

Int type.

Examples:

```
second('2014-09-01 12:30:45')=45
second('12:30:45')=45
```

```
second('20140901123045')=null
```

CURRENT_TIMESTAMP

Function definition:

```
timestamp current_timestamp()
```

Usage:

Returns the current timestamp as a Timestamp-type value. The value is not fixed.

Return Value:

Timestamp type.

Examples:

```
select current_timestamp() from dual;--Returns '2017-08-03 11:50:30.
661'
```

ADD_MONTHS

Function definition:

```
string add_months(string startdate, int nummonths)
```

Usage:

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'.

Examples:

```
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

Function definition:

```
string last_day(string date)
```

Usage:

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’.

Examples:

```
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

Function definition:

```
string next_day(string startdate, string week)
```

Usage:

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’.

Examples:

```
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

Function definition:

```
double months_between(datetime/timestamp/string date1, datetime/
timestamp/string date2)
```

Usage:

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:

Double type.

- When date1 is later than date2, the returned value is positive. When date2 is later than date1, the returned value is negative.
- When date1 and date2 correspond to the last days of two months, the returned value is an integer representing the number of months. Otherwise, the formula is (date1 - date2)/31.

Examples:

```
months_between('1997-02-28 10:30:00', '1996-10-30')=3.9495967741935485
months_between('1996-10-30','1997-02-28 10:30:00' )=-3.9495967741
935485
months_between('1996-09-30','1996-12-31')=-3.0
```

3.10.3 Window Functions

In MaxCompute SQL, window function can be used to analyze and process work flexibly. Window function can only appear in 'select' clause. You are not allowed to use nested window function and aggregate function in window function. It cannot be used with the same level aggregation function together.

Currently, in a MaxCompute SQL statement, you can use up to 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** is used to specify 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 (exceeding 5,000,000 rows are not advised though) ; otherwise, an error is reported at runtime.
- The clause **order by** is used to specify how the data is ordered in a window.
- In **windowing_clause** part, you can use rows to specify window open way. Two ways 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 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. You can use rows method to specify window range on condition that you have specified 'order by' clause.

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:

Calculate the total number of retrieved rows.

Parameter description:

- **expr**: any data type. When it is NULL, this row is not involved in count. If the 'distinct' keyword is specified, it indicates taking the unique count value.
- **partition by [col1, col2…]**: Specify 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 current window. If 'order by' clause is specified, the return

result is ordered according to specified sequence and the value is a cumulative count value from start row to current row in current window.

Return value:

Bigint type.

 **Note:**

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

Example:

Suppose that the table ‘test_src’ is existent 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 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 current window.
```

AVG

Function definition:

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

Usage:

Calculate the average.

Parameter description:

- **`distinct`**: if the keyword 'distinct' is specified, it indicates taking average of 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 operation. If it is another data type, an exception indicated.
  - If this value is NULL, this row is not involved in the calculation.
  - If the data type is Boolean, it is not allowed to be involved in the calculation.
- **`partition by [col1, col2...]`**: specified columns 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 current window. If 'order by' clause is specified, the return result is ordered according to specified sequence and returns the cumulative average from the start row to current row in current window.

Return value:

Double type.

Note:

If the keyword 'distinct' has been 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:

Calculate 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 unique value.
- **`partition by [col1, col2…]`**: specified 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 current window. If 'order by' clause is specified, the return result

is ordered according to specified sequence and return the maximum value from start row to current row in current window.

Return value:

the same type with 'expr'.

Note:

If the keyword 'distinct' has been 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:

Calculate the minimum value of the column.

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 minimum value of a unique value.
- **partition by [col1, col2…]**: specified 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 current window. If 'order by' clause is specified, the return result is ordered according to specified sequence and return the minimum value from start row to current row in current window.

Return value:

the same type with 'expr'.

Note:

If the keyword 'distinct' has been 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:

Calculate 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 will be converted to the Array of Double by default .
- **partition by [col1, col2…]**: specified 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:

Calculation population standard deviation.

Parameter description:

- **expr**: Double type.
  - If the input is ‘string’ or ‘bigint’ type, it is converted to ‘double’ type and involved in operation . If it is another data type, an exception is indicated.
  - If the input value is ‘NULL’, this row is ignored.
  - If the keyword ‘distinct’ is specified, it indicates calculating the population standard deviation of unique value.
- **partition by [col1, col2..]**: specified 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 current window. If ‘order by’ clause is specified, the

return result is ordered according to specified sequence and return the population standard deviation from start row to current row in 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’ has been specified, the ‘order by’ clause cannot be used.
- **Stddev** function also has an alias function named **stddev_pop**, whose usage is the same as **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 involved in operation . If it is another data type, an exception is indicated.
  - If the input value is NULL, this row is ignored.
  - If the keyword ‘distinct’ is specified, it indicates calculating the sample standard deviation of unique value.
- **partition by [col1, col2..]**: specified 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 current window. If ‘order by’ clause is specified, the return result is ordered according to specified sequence and return the sample standard deviation from start row to current row in current window.

Return value:

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

Note:

If the keyword ‘distinct’ has been 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:

Calculate the sum of elements.

Parameter description:

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

Calculate dense rank. The data in the same row of col2 has the same rank.

Parameter description:

- **`partition by [col1, col2..]`**: specified columns to use window function.
- **`order by col1[asc|desc], col2[asc|desc]`**: specify the value which the rank is based on.

Return value:

Bigint type.

Example:

Suppose that 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:

Calculate the rank. The ranking of the same row data with col2 drops.

Parameter description:

- **`Partition by [col1, col2..]`**: specify columns to use window function.
- **`Order by col1[asc|desc], col2[asc|desc]`**: specify the value which the rank is based on.

Return value:

Bigint type.

Example:

Suppose that 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]…]])
```

The command description is as follows:

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' oversteps the boundary. It is a constant and default is null.
- **partition by [col1, col2..]**: specify columns to use window function.
- **order by col1[asc|desc], col2[asc|desc]**: specify the order method for return result.

Return Value:

Returns the same with 'expr'.

LEAD

The command format is as follows:

```
lead(expr , Bigint offset, default) over(partition by [col1, col2…]
[order by [col1[asc|desc], col2[asc|desc]…]])
```

The command description is as follows:

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 oversteps the boundary. It is a constant.
- **partition by [col1, col2..]**: specify columns to use window function.
- **order by col1[asc|desc], col2[asc|desc]**: specify the order method for return result.

Return Value:

Returns the same with expr.

For 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

The command format is as follows:

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

The command description is as follows:

Calculate relative ranking of a certain row in a group of data.

Parameter description:

- **`partition by [col1, col2..]`**: specify columns to use window function.
- **`order by col1[asc|desc], col2[asc|desc]`**: specify the value which the ranking is based on.

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

The command format is as follows:

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

The command description is as follows:

This function is used to calculate the row number, beginning from 1.

Parameter description:

- **`partition by [col1, col2..]`**: specify columns to use window function.
- **`order by col1[asc|desc], col2[asc|desc]`**: specify the order method for return result.

Return Value:

Returns the Bigint type.

For example:

Suppose that 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

The command format is as follows:

```
boolean cluster_sample([Bigint x, Bigint y])
over(partition by [col1, col2..])
```

The command description is as follows:

This function is 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 value is 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 (that is to say, if y parts records exist, return value is true). If y is NULL, return NULL.
- **`partition by [col1, col2]`**: specify columns to use window function.

Return Value:

Returns the Boolean type.

For 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. 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 suggested:

```
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

The command format is as follows:

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

The command description is as follows:

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.

The example is as follows:

Assume 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 |
```

3.11 UDF

3.11.1 UDF Summary

MaxCompute provides many built-in functions to meet the computing requests of a user and a user can also create user-defined functions to meet different computing needs. A User Defined Function (UDF) is similar to an ordinary Built-in Function. MaxCompute provides many built-in functions to meet the computing requests of you and you can also create user-defined functions to meet different computing needs. UDF is similar to an ordinary *Built-in Function* .

If you use *Maven* to search “odps-sdk-udf” from *Maven* to get different versions of Java SDK. The configuration information is shown as follows:

```
<dependency>
    <groupId>com.aliyun.odps</groupId>
    <artifactId>odps-sdk-udf</artifactId>
   <version>0.20.7-public</version>
```

```
</dependency>
```

In MaxCompute, you can expand two kinds 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 (User-defined 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 UDF can 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 Group By clause together. For more information, see *Aggregation Functions*.

Note:

- UDF stands for the set of use-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 not enough with an UDF involved in the SQL statement, configure `set odps.sql.udf.joiner.jvm.memory=xxxx;` to solve it. The reason is that the amount of data is too large and there is a data skew, so that the memory size occupied by task 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 about authorization, 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.

3.11.2 Java UDF

MaxCompute UDF includes three types: UDF, UDAF, and UDTF.

Parameter and return value type

The data types of UDF supported by MaxCompute SQL include: Basic 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 such as tinyint, smallint, int, float, varchar, binary, and timestamp by a 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 does not yet support new basic types.
- JAVA UDF uses three complex data types — 'array', 'map', and 'struct':
  - UDTF specify 'signature' by @Resolve annotation, for example, `@Resolve("array<string>,struct<a1:bigint,b1:string>,string->map<string,bigint>,struct<b1:bigint>")Sinliterature, such as: @ resolve ("array <string>, struct <A1: bigint, b1: String>, string-> Map <string, bigint>, struct <B1: bigint> ")`.
  - UDF maps the input and output types by 'signature' of the evaluation method. In this case, see the mapping between the MaxCompute type and the Java type. In the mapping, 'array' corresponds to 'java.util.List'; 'map' corresponds to 'java.util.Map'; and 'struct' corresponds to 'com.aliyun.odps.data.Struct'.
  - UDAF does not yet support.

> **Note:**
>
> - By 'com.aliyun.odps.data.Struct', the field name and field type cannot be seen in the reflection; therefore, @Resolve annotation is needed. That is to say, if 'struct' is needed in UDF, @Resolve annotation must be added to the UDF class. This annotation only affects the reload of the parameter or returned values that include 'com.aliyun.odps.data.Struct'.

- *Currently, only one @Resolve annotation can be provided in one class. Therefore, only one reload of parameters or returned values that carry 'struct' is allowed in a UDF.*

MaxCompute data types and Java data types correspond as follows:

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 implemented. The 'evaluate' method must be non-static public method. The

parameter type and return value type of Evaluate method is considered as UDF signature in SQL . This means that the user can implement multiple evaluate methods in UDF. To call UDF, the framework matches correct evaluate method according to the parameter type called by UDF.

Classes with the same class name but different functional logic would better appear 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, it causes udf execution behavior is inconsistent or even failed to compile.

UDF example:

```
package org.alidata.odps.udf.examples;
import com.aliyun.odps.udf.UDF;

public final class Lower extends UDF
{ public String evaluate(String s) {
if (s == null) { return null; }
return s.toLowerCase(); }
```

You can achieve UDF initialization and end 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*.

UDAF

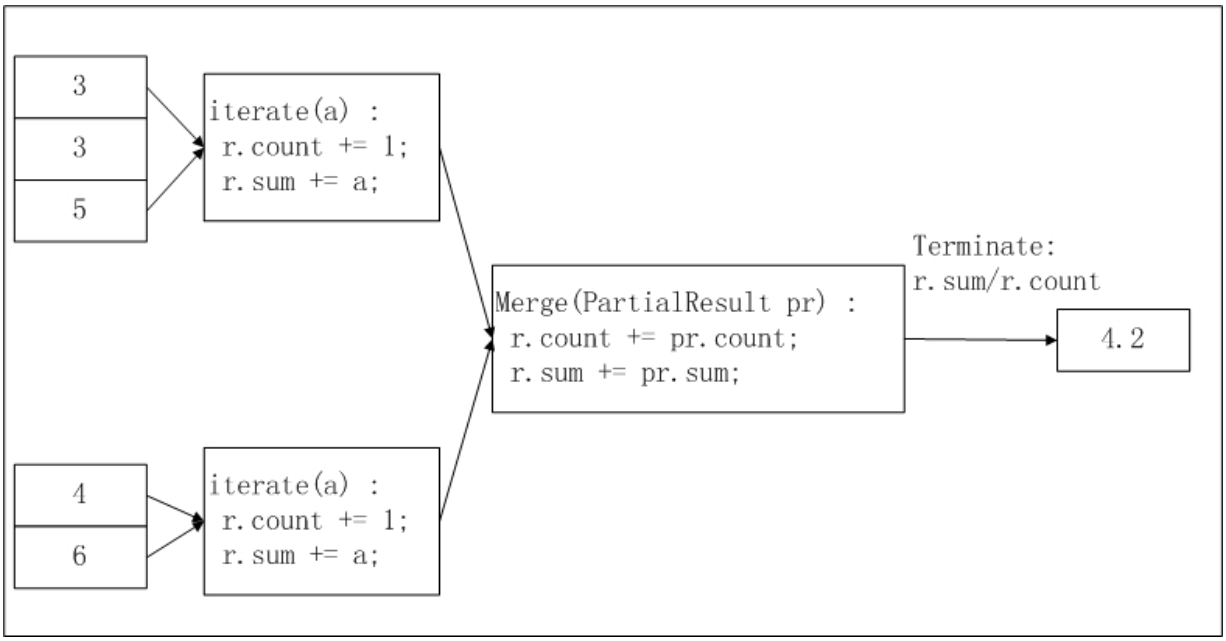
To implement Java UDAF, you must inherit the class 'com.aliyun.odps.udf.UDAF' and the following interfaces must be implemented:

```
public abstract class Aggregator implements ContextFunction {
  @Override
  public void setup(ExecutionContext ctx) throws UDFException {


  @Override
  public void close() throws UDFException {


   * Creat aggregation Buffer
   * @return Writable aggregation 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 needs to 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 image displayed preceding, the input data is sliced according to certain size (For the description of slicing, see *MapReduce*). The size of each slice is suitable for a worker completed in appropriate time. This slice size needs to be configured by the user manually.

The calculation process of UDAF is divided into two stages:

- In the first stage, each worker counts the data quantity and total sum in a slice. You can take the data quantity and total sum in each slice as an intermediate result.
- In the second stage, 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.

The following is a UDAF encoding example to calculate 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, because 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 needs to 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 needs to 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 needs to 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.

Next a UDTF program example is shown 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 UDF. For more information, see *Java UDF Development*.

In SQL you can use this UDTF as 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 |
```

UDTF Instructions for Use

In SQL, the common use method of UDTF is shown as follows:

```
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 limitations:

- 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, you can read MaxCompute *Resources*. Next introduce how to read MaxCompute resources by using UDF:

1. Write the UDF program and compile it successfully. Then export it as a 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 tabkes: table_resource1, table_resource2 and the physical table tmp1 in MaxCompute. Insert corresponding data into the tables.

5. Run this UDF:

```
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 a 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 has the @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.

Note:
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 can support all data types, including array, map, struct, and other complex types.

Note:

- MaxCompute's add jar command permanently creates a resource in the project, so you must 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;` in front of the SQL statement, and submit it with SQL statement.
- When using compatible Hive UDFs, you must pay attention to *Java sandbox* restrictions of MaxCompute.

3.11.3 Python UDF

The Maxcompute UDF consists of UDF, UDAF, and UDTF functions, this article focuses on how to implement these three functions through 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 all be implemented by 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:

- 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

- unicodedata
- _weakref
- cPickle

- Some modules have limited functionality. 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 that is currently supported by the python UDF SQL data types 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 need to specify the function signature by adding a decorator to the UDF class.

The function signature is specified by a string. The syntax is as below:

```
Arg_type_list "-> type_list
Arg_type_list: db_list | '*' |''
Db_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.

The legitimate 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 will be checked out, and an error is returned. The execution would then be banned. 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 The 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, just need to define a new-style Class, and implements the evaluate method. The example is as follows:

```
From ODPS. UDF import quilate

@ Quilate ("bigint, bigint-> bigint ")
Class myplus (object ):

   Def evaluate (self, arg0, arg1 ):
       If none in (arg0, arg1 ):
```

```
            Return none
        Return arg0 + arg1
```

Note:
The Python UDF must specify the function signature through ‘annotate’.

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 mutable Object (such as list, dict), and the size of the buffer should not increase with the amount of data, in the case of limit, Buffer The size after Marshal should 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’. Merge 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:

```
@ Attenuate ('double-> Doubles ')
Class Average (baseudag ):

    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 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, etc.

- BaseUDTF.init(): the initialization method. If the inherited class implements this method, then it must call the initialization method of the basic class ‘super(BaseUDTF, self).init()’ from the beginning. .

  The ‘init’ method will only be called once during the entire UDTF lifecycle; that is, before the first record is processed. When the UDTF needs to save the internal state, all states can be initialized in this method.
- Baseudtf. Process ([ARGs,...]): This method by maxcompute 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 quilate
From ODPS. UDF import baseudtf


@ Attenuate ('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:

Python 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.

Reference Resources

Python UDF can reference resource files through the ‘odps.distcache’ module. Currently, referencing file resources and table resources is supported.

- odps.distcache.get_cache_file(resource_name)
  - Returns the resource content for the specified name. resource_name: ‘str’ type, corresponding to the existing resource name in the current project. If the resource name is invalid or has no responding resources, return an error.
  - The return value is file-like object The caller is obliged to 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:

```
@ Attenuate ('bigint-> string ')
Class distcacheexample (object ):

Def _ init _ (Self ):
    Cache_file = Porter ')
    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, return an error.
  - Returned value: returned value is ‘generator’ type. The caller obtain 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 quilate
From ODPS. distcache import fig

@ Attenuate ('-> string ')
Class maid (object ):
    Def _ init _ (Self ):
        Self. Records = List ('udf _ test '))
        Self. Counter = 0
```

```
        Self. Ln = Len (self. Records)

    Def evaluate (Self ):
        if self.counter > self.ln - 1:
            Return none
        Ret = self. Records [self. Counter]
        Self. Counter + = 1
        Return STR (RET)
```

3.12 Appendix

3.12.1 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

Escape	Character
\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'.

3.12.2 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%cd' like 'ab\\%%' -- true
```

Note:

MaxCompute SQL only supports the UTF-8 character set. If the data is encoded in another format, it is possible that the calculation result is not correct.

3.12.3 Regular Expression

The regular expressions in MaxCompute SQL use the PCRE standard, matched by characters. The metacharacter 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

Metacharacter	Description
	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	[][!” [][!”#$%&’()*+,./:;<=>? @\^_ `{|}~-]

POSIX Character Group	Description	Range
[[: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 |
```

3.12.4 Reserved Words

This document shows all reserved words in MaxCompute SQL.

Note:

- 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 MapReduce

4.1 Summary

4.1.1 MapReduce

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. Map/Reduce is implemented in the same manner as the MaxCompute native interface.
- *Hadoop compatible version*: highly compatible with *Hadoop MapReduce* , but not compatible with MaxCompute native interface and *MR2*.

The above three versions are basically the same in the *basic concepts*, *Job submission*, *input and output*, and *resource*, and the difference is the Java SDK. This document 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 example is as follows:

- Search: web crawl, flip index, PageRank.
- Web access log analytics:
  - Analize and mine the web access, shopping behavior characteristics to achieve personalized recommendation.
  - Analyze user's access behavior.
- Statistics and analysis for text:
  - The Wordcount and TFIDF analysis of Mo Yan novels.
  - Reference analysis and statistics of academic papers and patent documents.
  - Wikipedia data analysis, etc.

- 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, etc.
- 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 Process

The processing data process of MapReduce is divided into two stages: Map and Reduce . You execute Map first, and then Reduce. The processing logic of Map and Reduce is defined by a user, but must comply with the MapReduce framework protocol. The processing process is as follows:

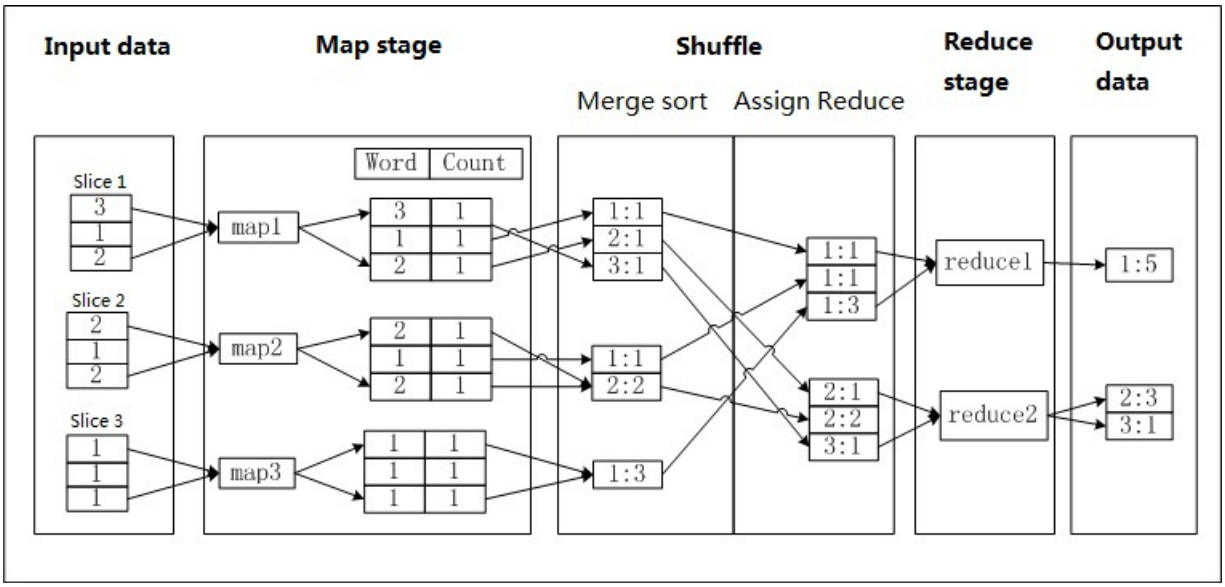
1. Before executing Map, the input data must be **sliced**, that is, input data is divided into blocks of equal size. Each block is processed as the input of a single Map Worker, so that multiple Map Workers can work simultaneously.
2. After the slice is split, multiple Map Worker can work together. Each Map Worker performs computing after reading the data and output the result to Reduce. Because Map Worker outputs the data, it needs to specify a key for each output record. The value of this Key determines which Reduce Worker the data is 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 Reduce in MaxCompute. This processing is generally called **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. Multiple records of the same key then become a value through Reduce processing.

Note:

A brief introduction to the MapReduce framework is provided above. For more details, see relevant documents.

The following example uses WordCount to explain the stages of MaxCompute MapReduce.

Assume there is a text named ‘a.txt’, where each row is indicted by a number, and the frequency of appearance of each number needs to be counted. The number in the text is called ‘Word’ and the number appearance occurrence is called Count. To complete this function though MaxCompute MapReduce, the following figure details the steps required:



Procedure:

1. First, text is sliced and the data in each slice is input 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. and 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). Then 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 combiner sorting.
4. In the later actions of Shuffle, data is transmitted to Reduce. Reduce Worker sorts the data based on Key value again after receiving data.
5. At the time of processing data, each Reduce Worker adopts that same logic as that of Combiner by accumulating Count with same Key value (Word value) to get the output
6. result.

Note:

Because data in MaxCompute are 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.

4.1.2 Extended MapReduce

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 needs to be written to a disk. The following Map task is an example of a task that was only to read data, prepared for subsequent Shuffle stage, but which actually resulted in redundant I/O operations.

The calculation scheduling logic of MaxCompute can support more complex programming paradigm. In the preceding scenario, the next Reduce operation can be executed after Reduce operation and inserting a Map operation is not necessary. In this way, MaxCompute provides an extensional MapReduce model, that is, numerous Reduce operations can follow a Map operation, such as Map>Reduce> Reduce.

Hadoop Chain Mappper/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 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 include that 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 the way map/reduce functions are written. The main difference is in performance. For more information, see *Extended MapReduce example*.

4.1.3 Open-source MapReduce

MaxCompute offers a set of native MapReduce programming models and interfaces. The inputs and outputs for these interfaces are MaxCompute tables, and data is organized for processing in record format.

However, MaxCompute APIs differ significantly from APIs for the Hadoop framework. Previously , if you wanted to migrate your Hadoop MapReduce jobs to MaxCompute, you needed to first 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 do not need to modify or 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. This means that, without modifying the code, you can specify configurations to directly run original Hadoop MapReduce JAR packages on MaxCompute. You can download the development plug-in to get *started*. This plug-in is currently in the testing stage and 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.

Note:

- For more information on open-source compatibility, see *Open-source SDK compatibility*.
- For more information about the Hadoop MapReduce SDK, see *the Official MapReduce documentation*.

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 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 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 text file to be imported is as follows:

```
hello maxcompute
hello mapreduce
```

you can use the `Tunnel` command at 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"
```

Configuration item descriptions:

The configuration is a JSON file that describes the mapping relationships between HDFS files and 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, you must also 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, you must 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, you must 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. You can use a regular expression in 'fuzzy' mode to match the HDFS input path.

Job Submission

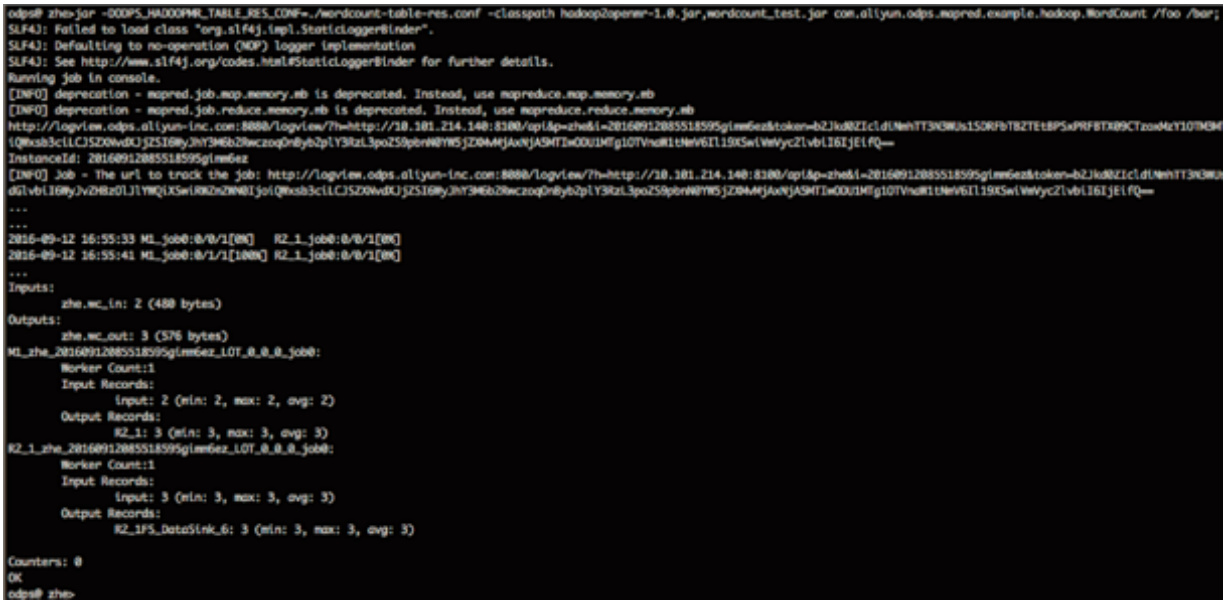
Use the MaxCompute command line tool odpscmd to submit jobs. For the installation and configuration of MaxCompute command line tool, see *the Console*. In odpscmd, run the following command:

```
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;
```

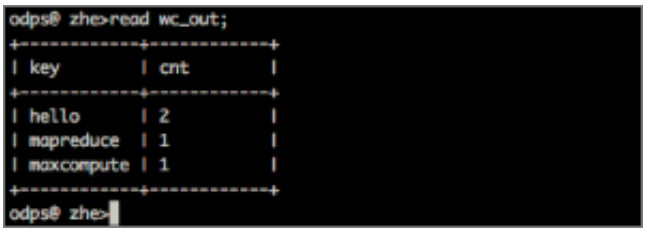
Note:

- 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, you must 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, you must make the relevant changes when specifying the configuration and -classpath.

The running process is shown in the following figure:



After running the job, check the results table wc_out to verify that the job has been completed.



4.2 Function Introduction

4.2.1 Command

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 need to 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 local mode can be defined.
- -l: run MapReduce job in local mode, mainly used for program debugging.

User can specify the configuration file 'JobConf' by option '-conf'. This file can modify the JobConf settings in the SDK.

An example of a configuration file 'JobConf' is as follows:

```
<configuration>
       <property>
          <name>import.filename</name>
          <value>resource.txt</value>
       </property>
    </configuration>
```

In the preceding example, the variable 'import.filename' is defined and its value is 'resource.txt'.

User can get this variable value through the JobConf interface in the MapReduce program. Alternatively, users can also get the value through the JobConf interface in the SDK. For a detailed example, see *Use Resource Example*.

Example:

```
add jar data\mapreduce-examples.jar;
    jar -resources mapreduce-examples.jar -classpath mapreduce-
examples.jar
        org.alidata.odps.mr.examples.WordCount wc_in wc_out;
    add file data\src.txt;
    add jar data\mapreduce-examples.jar;
    jar -resources src.txt,mapreduce-examples.jar -classpath data\
mapreduce-examples.jar
        org.alidata.odps.mr.examples.WordCount wc_in wc_out;
    add file data\a.txt;
    add table wc_in as test_table;
    add jar data\work.jar;
    jar -conf odps-mapred.xml -resources a.txt,test_table,work.jar
        -classpath data\work.jar:otherlib.jar
```

```
        -D import.filename=resource.txt org.alidata.odps.mr.examples.
WordCount args;
```

4.2.2 Basic Conception

Map/Reduce

Map and Reduce support corresponding map/reduce method, setup method and cleanup method . The setup method is called before the map/reduce method, and each worker only 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 sort columns as Group columns, but user-defined Group comparator is not supported. Sort columns are used to sort your data while Group columns are used for secondarySort.

For more information, see *SecondarySort Example*.

Partition

Supports setting the partition column and customized partitioner. Partition columns have a higher priority than customized partitioners.

The partitioner is used to distribute the output data on Map terminal to different Reduce Workers according to Hash logic.

Combiner

The Combiner function combines adjacent records in Shuffle stage. You can choose whether to use Combiner according to different business logic.

Combiner helps 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 Map terminal.

For more information, see *WordCount Code Examples*.

4.2.3 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 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 and the default output does not need label. No output is not allowed.

For more input and output examples, see *Program Examples*.

4.2.4 Resource

You can read MaxCompute resources in Map/Reduce. Any Worker of Map/Reduce can load resources to memory for you to apply in code use.

For more information, see *Use Resource Example*.

4.2.5 Local run

Basic stages Introduction

Local run prerequisite: By setting `-local` parameter in jar command, user can simulate MapReduce running process on the local to continue 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 ‘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 at next running time, and do not repeat the downloading.

Differences between running locally and running distributed environments

In the local operation course, multiple Map and Reduce workers are still started to process data. But these workers are not running concurrently and followed by serial running.

In addition, this simulation process and real distributed operation have the following differences:

- A restriction for row number of input table exists: now, up to 100 rows of data can be downloaded.
- Usage of resource: in distributed environment, MaxCompute limits the size of referenced resource. For more information, see *Application Restriction*. Note that in local running environment, the resource size is no limitation.
- Security restriction: MaxCompute MapReduce and UDF program running in a distributed environment are limited by *Java Sandbox*. Note that in local operations the restriction does not exists.

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 a colon ‘:’, and columns and columns are separated by a comma ‘,’. In the front of <__schema__> file, the Project name and Table name must be declared, such as `project_name.table_name`, and separated by comma and column definition. `project_name.table_name,col1_name:col1_type,col2_name:col2_type,……`
- The file ‘data; indicates table data. The column quantity and corresponding data must comply with the definition in *schema_*, that is, extra columns and missing columns are not allowed.

  The content of *Cite Left_schema_ Cite Right*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:
  Note that the function to download 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 *Cite Left_schema_ Cite Right* 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 *Cite Left_schema_ Cite Right*. The file 'data' is a result data file generated after local operation.

Note:

- Users can also edit *Cite Left_schema_ Cite Right* file and 'data' and then place these two files into the corresponding table directory.
- When running on the local, the client detects that the table directory already exists, and does not download the information of this table from MaxCompute. The table directory on the local can be a table that does not exist in MaxCompute.

4.3 Program Example

4.3.1 WordCount Sample

Preparation

1. Prepare the JAR package of the test program. Assume the package is named mapreduce-examples.jar. 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 data imported into the wc_in table is 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
```

Result

The job is successfully completed.

```

| 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();

        @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);
        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);
        JobClient.runJob(job);
```

4.3.2 MapOnly Sample

For MapOnly jobs, Map directly sends < Key, Value > pairs to tables on MaxCompute. You only need to specify the output table but do not need to specify the Key/Value metadata to be output by Map.

Preparation

1. Prepare the JAR package of the test program. Assume the package is named mapreduce-examples.jar.
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. Run tunnel to import data.

   ```
   tunnel upload data wc_in;
   ```

   The data imported into the wc_in table is as follows:

   ```
   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
```

Result

The output table ‘wc_out’ contains the following content:

```
| 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);
          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);
          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);
          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);
        job.setNumReduceTasks(0);
        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]);
          if (options.contains("setup")) {
            job.setBoolean("option.mapper.setup", true);

          if (options.contains("map")) {
            job.setBoolean("option.mapper.map", true);

          if (options.contains("cleanup")) {
            job.setBoolean("option.mapper.cleanup", true);


        JobClient.runJob(job);
```

4.3.3 Multi-input and Output

Preparation

1. Prepare the 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 multi-table 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;
```

Result

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 {
          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");


      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);
```

4.3.4 Multi-task Sample

Preparation

1. Prepare the 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 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;
```

Result

The output table ‘mr_multijobs_out’ contains the following content:

```
| 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;
```

```
          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);
          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"));
        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);
          decJob.setNumReduceTasks(0);
          RunningJob rJob = JobClient.runJob(decJob);
          iterCount--;
          if (rJob.getCounters().findCounter("multijobs", "value").
getValue() == 0) {
            break;


        if (iterCount ! = 0) {
```

```
            throw new IOException("Job failed.")
```

4.3.5 Secondary Sort Sample

Preparation

1. Prepare the 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;
```

Result

The content 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);
```

4.3.6 Resource Sample

Preparation

1. Prepare the jar package of test program. Suppose that the package is named “mapreduce-examples.jar”, *The local storage path is data\resources*.
2. Prepare the test table and 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;
```

Result

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);
        job.set("import.filename", args[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);
```

In fact, user has several methods to set up JobConf:

- Set it through JobConf interface in SDK. This example above is through this method and this method is of the highest priority.

- In jar command lines, specify new JobConf file through the parameter **-conf**. This method is of the lowest priority.

4.3.7 Counter Sample

Preparation

1. Prepare the JAR package of the test program. Assume the package is named “mapreduce-examples.jar”, *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 contents of data file imported into the table “wc_in”:

   ```
   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
```

Result

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);

        Counters counters = rJob.getCounters();
        long m = counters.findCounter(MyCounter.MAP_TASKS).getValue();
        long r = counters.findCounter(MyCounter.REDUCE_TASKS).getValue
();
        long total = counters.findCounter(MyCounter.TOTAL_TASKS).
getValue();
        System.exit(0);
```

4.3.8 Grep Sample

Preparation

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;
```

Result

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);
        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);
        JobConf sortJob = new JobConf();
        sortJob.setMapperClass(InverseMapper.class);
        sortJob.setReducerClass(IdentityReducer.class);
        sortJob.setMapOutputKeySchema(SchemaUtils.fromString("count:
bigint"));
        sortJob.setMapOutputValueSchema(SchemaUtils.fromString("word:
string"));
        InputUtils.addTable(TableInfo.builder().tableName(args[1]).
build(), sortJob);
        OutputUtils.addTable(TableInfo.builder().tableName(args[2]).
build(), sortJob);
        sortJob.setNumReduceTasks(1); // write a single file
        sortJob.setOutputKeySortColumns(new String[] { "count" });
        @SuppressWarnings("unused")
        RunningJob rjSort = JobClient.runJob(sortJob);
```

4.3.9 Join Sample

MaxCompute MapReduce framework does not support JOIN, however, you can implement data join in your Map/Reduce function.

Assume that table mr_join_src1(key bigint, value string) must be joined with mr_join_src2(key bigint, value string). The output table is mr_join_out (key bigint, value1 string, value2 string). value1 is value in mr_join_src1 and value2 is value in mr_join_src2.

Preparation

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 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 data.

   ```
   tunnel upload data1 mr_Join_src1;
   tunnel upload data2 mr_Join_src2;
   ```

   The data imported into the mr_join_src1 table is as follows:

   ```
   1,hello
   ```

```
 2,odps
```

The data imported into the mr_join_src2 table is as follows:

```
1,odps
3,hello
4,odps
```

Procedure

Run Join in odpscmd.

```
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;
```

Result

The output table mr_join_out contains the following content:

```
| 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();

        @Override
        public void reduce(Record key, Iterator<Record> values,
TaskContext context)
            throws IOException {
          long k = key.getBigint(0);
          List<Object[]> leftValues = new ArrayList<Object[]>();
          while (values.hasNext()) {
            Record value = values.next();
            long tag = (Long) key.get(1);
            if (tag == 0) {
              leftValues.add(value.toArray().clone());
            } else {
              for (Object[] leftValue : leftValues) {
                int index = 0;
                result.set(index++, k);
                for (int i = 0; i < leftValue.length; i++) {
                  result.set(index++, leftValue[i]);

                for (int i = 0; i < value.getColumnCount(); i++) {
                  result.set(index++, value.get(i));

                context.write(result);




      public static void main(String[] args) throws Exception {
        if (args.length ! = 3) {
          System.err.println("Usage: Join <input table1> <input table2
> <out>");
          System.exit(2);

        JobConf job = new JobConf();
        job.setMapperClass(JoinMapper.class);
        job.setReducerClass(JoinReducer.class);
        job.setMapOutputKeySchema(SchemaUtils.fromString("key:bigint,
tag:bigint"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("value:
string"));
```

```
        job.setPartitionColumns(new String[]{"key"});
        job.setOutputKeySortColumns(new String[]{"key", "tag"});
        job.setOutputGroupingColumns(new String[]{"key"});
        job.setNumReduceTasks(1);
        InputUtils.addTable(TableInfo.builder().tableName(args[0]).
label("left").build(), job);
        InputUtils.addTable(TableInfo.builder().tableName(args[1]).
label("right").build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[2]).
build(), job);
        JobClient.runJob(job);
```

4.3.10 Sleep Sample

Preparation

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, 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;
```

Result

The job ran 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 {
    @Override
    public void setup(TaskContext context) throws IOException {
      try {
```

```
        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);
    job.setNumReduceTasks(0);
    job.setNumMapTasks(1);
    job.set(SLEEP_SECS, args[0]);
    JobClient.runJob(job);
```

4.3.11 Unique Sample

Preparation

1. Prepare the 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 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 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;
```

Result

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];

        // 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);
```

4.3.12 Sort Sample

Preparation

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 imported into the table ss_in:

```
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;
```

Result

The content of 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);
        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.")
```

4.3.13 Partition

The following example takes Partition as input and output.

Example 1:

```
     public static void main(String[] args) throws Exception {
     JobConf job = new JobConf();

     LinkedHashMap<String, String> input = new LinkedHashMap<String,
String>();
     input.put("pt", "123456");
     InputUtils.addTable(TableInfo.builder().tableName("input_table").
partSpec(input).build(), job);
     LinkedHashMap<String, String> output = new LinkedHashMap<String,
String>();
     output.put("ds", "654321");
     Outpututils. addtable (tableinfo. builder (). tablename ("
output_table "). partspec (output ). build (), job );
     JobClient.runJob(job);
```

Example 2:

```
    package com.aliyun.odps.mapred.open.example;

      public static void main(String[] args) throws Exception {
        if (args.length ! = 2) {
          System.err.println("Usage: WordCount <in_table> <out_table
>");
          System.exit(2);

        JobConf job = new JobConf();
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(SumCombiner.class);
        job.setReducerClass(SumReducer.class);
        job.setMapOutputKeySchema(SchemaUtils.fromString("word:string
"));
        job.setMapOutputValueSchema(SchemaUtils.fromString("count:
bigint"));
       Account account = new AliyunAccount("my_access_id", "
my_access_key");
        Odps odps = new Odps(account);
        odps.setEndpoint("odps_endpoint_url");
        odps.setDefaultProject("my_project");
        Table table = odps.tables().get(tblname);
        TableInfoBuilder builder = TableInfo.builder().tableName(
tblname);
        for (Partition p : table.getPartitions()) {
          if (applicable(p)) {
            LinkedHashMap<String, String> partSpec = new LinkedHashMap
<String, String>();
            for (String key : p.getPartitionSpec().keys()) {
              partSpec.put(key, p.getPartitionSpec().get(key));

            InputUtils.addTable(builder.partSpec(partSpec).build(),
conf);
```

```
        OutputUtils.addTable(TableInfo.builder().tableName(args[1]).
build(), job);
        Jobclient. runjob (job );
```

Note:

- The preceding example combines the MaxCompute SDK and MapReduce SDK to achieve a MapReduce task.
- The code cannot be compiled and is only an example of main functions.
- The Applicable function is user logic that determines whether the Partition can be used as the input of MapReduce job.

4.3.14 Pipeline Sample

Preparation

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 contents of data file imported into the table wc_in, 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;
```

Result

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();
        job.setPipeline(pipeline);
        job.addInput(TableInfo.builder().tableName(args[0]).build());
        job.addOutput(TableInfo.builder().tableName(args[1]).build());
        job.submit();
```

```
        job.waitForCompletion();
        System.exit(job.isSuccessful() == true ? 0 : 1);
```

4.4 Java SDK

4.4.1 Java SDK

This section introduces common MapReduce interfaces.

The users who use Maven can search “odps-sdk-mapred” from *Maven Library* to get the required Java SDK (available in different versions). The configuration is shows 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 needs to inherit from the 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 job running process.
JobConf	Describes configuration of a MapReduce task. The JobConf object is generally defined in 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)	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)	reduce method, process record of input table.
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()	Get output table information.
Record createOutputRecord()	Create the record object of the default output table.
Record createOutputRecord(String label)	Create the record object of the output table with provided label.
Record createMapOutputKeyRecord()	Create the record object of Key output by Map.
Record createMapOutputValueRecord()	Create the record object of Value output by Map.
void write(Record record)	It writes record to default output and is used for writing output data by Reduce client, and can be called on the Reduce client multiple times.
void write(Record record, String label)	It writes record to the given label output and is used for writing output data by Reduce client,

Interface	Description
	and can be called on the Reduce client multiple times.
void write(Record key, Record value)	Map writes record to intermediate result. It can be called in Map function and called on the Map client multiple times.
BufferedInputStream readResourceFileAsStream(String resourceName)	Read file type resource.
Iterator<Record > readResourceTable(String resourceName)	Read table type resource.
Counter getCounter(Enum<? > > name)	Get the Counter object with provided name.
Counter getCounter(String group, String name)	Get the Counter object with provided group name and name.
void progress()	Report heartbeat information to the MapReduce framework. It reports heartbeat information to 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.

Note:

MaxCompute TaskContext interface provides the progress function, however, this function is to prevent the Worker being killed as it is running 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. MaxCompute The default timeout schedule of MaxCompute MapReduce Worker is 10 minutes (system default, not be controlled by 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 you call the progress interface regularly in Mapper/Reducer functions, to prevent being killed by the framework.

JobConf

Main function interfaces are as follows.

Interface	Description
void setResources(String resourceNames)	Declare resources this job uses. Declare resources used in this job. Only the declared resource can be read by TaskContext object during Mapper/Reducer running process.
void setMapOutputKeySchema(Column[] schema)	Set the Key attribute output from Mapper to Reducer.
void setMapOutputValueSchema(Column[] schema)	Set the Value attribute output from Mapper to Reducer.
void setOutputKeySortColumns(String[] cols)	Set key sort columns output from Mapper to Reducer.
void setOutputGroupingColumns(String[] cols)	Set Key grouping columns.
void setMapperClass(Class<? extends Mapper > theClass)	Set Mapper function of the job.
void setPartitionColumns(String[] cols)	Set the partition column specified in the job. The default is all columns of Key output by Mapper.
void setReducerClass(Class<? extends Reducer theClass)	Set Reducer of the job.
void setCombinerClass(Class<? extends Reducer theClass)	Set 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)	Set the size of input slice. Unit: MB. The default value is 640.
void setNumReduceTasks(int n)	Set the number of Reducer tasks. The default is 1/4 of Mapper tasks.
void setMemoryForMapTask(int mem)	Set the memory size of single Worker in Mapper task. Unit: MB. The default value is 2048.
void setMemoryForReduceTask(int mem)	Set the memory size of single Worker for Reducer task. Unit: MB. The default value is 2048.

 **Note:**

- Usually, GroupingColumns is included in KeySortColumns, while KeySortColumns and PartitionColumns are included in 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 sorted by KeySortColumns, input records are grouped as input groups of the reduce function sequentially, that is, 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)	Return immediately after submitting a MapReduce job in synchronous (blocking) mode.
static RunningJob submitJob(JobConf job)	Return immediately after submitting a MapReduce job in asynchronous (non-blocking) mode.

RunningJob

Main function interfaces are as follows.

Interface	Description
String getInstanceID()	Get instance ID for checking run log and job management.
boolean isComplete()	Check whether job is complete.
boolean isSuccessful()	Check whether job instance is successful.
void waitForCompletion()	Wait until job instance is complete. It is typically is used for jobs submitted is asynchronous mode.
JobStatus getJobStatus()	Check job instance status.
void killJob()	End the job.
Counters getCounters()	Get Counter information.

InputUtils

Main function interfaces are as follows.

Interface	Description
static void addTable(TableInfo table, JobConf conf)	Add table to task input. It can be called multiple times. The new added table is added to input queue in an append manner.
static void setTables(TableInfo [] tables, JobConf conf)	Add tables to task input.

OutputUtils

Main function interfaces are as follows.

Interface	Description
static void addTable(TableInfo table, JobConf conf)	Add table to task output. It can be called multiple times. Add the new added table to output queue in an append manner.
static void setTables(TableInfo [] tables, JobConf conf)	Add multiple tables to the task output.

Pipeline

Pipeline is the subject of *MR2* . A Pipeline can be constructed by Pipeline.builder. Pipeline 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 function of MapReduce, then the use of MR2 is similar.

Note:

- Specifically, we recommend that users can 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	Bigint	String	Double	Boolean	Datetime	Decimal
Java Type	Long	String	Double	Boolean	date	BigDecimal

4.5 MR Restrictions

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	2048M + 1024M	Yes	Memory occupied by a single map instance or reduce instance, including the framework memory (2

Restricted item	Value	Type	Configuration item	Default value	Configu le?	Description
			(reducer).jvm. mem			,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.
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.

Restricted item	Value	Type	Configuration item	Default value	Configu le?	Description
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.
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 .

Restricted item	Value	Type	Configuration item	Default value	Configu le?	Description
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.
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.

5 Java Sandbox

Java sandbox related restrictions of MaxCompute MapReduce and UDF programs running in distributed environments are as follows:

- Direct access to local files is not allowed. You can only access files by using interfaces provided by MaxCompute MapReduce/Graph in the following way:
  - 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:

Methods for accessing local files are as follows:

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 void final 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 void final 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, DomainComb
iner combiner)
    public DomainCombiner getDomainCombiner()
```

6 SDK

6.2 Python SDK

PyODPS is the Python SDK of MaxCompute. It supports basic actions on MaxCompute objects and the DataFrame framework for ease of data analysis on MaxCompute. For more information, see the *GitHub project* and the *PyODPS Documentation* that describes all interfaces and classes.

- For more information about PyODPS, see the *PyODPS community album*.
- Developers are welcome to participate in the ecological development of PyODPS. For more information, see *GitHub document*.
- You are welcome to submit the issue and merge request to accelerate PyODPS eco-growth. For more details, see *code*.
- DingTalk technology exchange group: **11701793**

Installation

PyODPS supports Python 2.6 and later versions. After installing PIP in the system, you only need to run `pip install pyodps`. The related dependencies of PyODPS are automatically installed.

Quick Start

Log on using your Alibaba Cloud primary account to initialize a MaxCompute entry, as shown in the following code:

```
from odps import ODPS
odps = ODPS('**your-access-id**', '**your-secret-access-key**', '**
your-default-project**',
            endpoint='**your-end-point**')
```

After completing initialization, you can operate tables, resources, and functions.

Project

A project is the basic unit of operation in MaxCompute, which is similar to a database.

Call `get_project` to obtain a project, as shown in the following code:

```
project = odps.get_project('my_project') # Obtain a project.
project = odps.get_project() # Obtain the default project.
```

Note:

- If parameters are not input, use the default project.
- You can call `exist_project` to check whether a project exists.
- A table is a data storage unit of MaxCompute.

Table Actions

Call `list_tables` to list all tables in the project, as shown in the following code:

```
for table in odps.list_tables():
    # Process each table
```

Call `exist_table` to check whether the table exists and call get_table to obtain the table.

```
>>> t = odps.get_table('dual')
>>> t.schema
odps.Schema {
  c_int_a                 bigint
  c_int_b bigint
  c_double_a double
  c_double_b double
  c_string_a string
  c_string_b string
  c_bool_a boolean
  c_bool_b boolean
  c_datetime_a datetime
  c_datetime_b datetime
}
>>> t.lifecycle
-1
>>> print(t.creation_time)
2014-05-15 14:58:43
>>> t.is_virtual_view
False
>>> t.size
1408
>>> t.schema.columns
[<column c_int_a, type bigint>,
 <column c_int_b, type bigint>,
 <column c_double_a, type double>,
 <column c_double_b, type double>,
 <column c_string_a, type string>,
 <column c_string_b, type string>,
 <column c_bool_a, type boolean>,
 <column c_bool_b, type boolean>,
 <column c_datetime_a, type datetime>,
 <column c_datetime_b, type datetime>]
```

Create a table's schema

Two initialization methods are provided:

- Initialize through table columns and optional partitions, as shown in the following code:

```
>>> from odps.models import Schema, Column, Partition
>>> columns = [Column(name='num', type='bigint', comment='the column
')]
```

```
>>> partitions = [Partition(name='pt', type='string', comment='the
partition')]
>>> schema = Schema(columns=columns, partitions=partitions)
>>> schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

- Although it is easier to call `Schema.from_lists` for initialization, annotations of columns and partitions cannot be directly set.

```
>>> schema = Schema.from_lists(['num'], ['bigint'], ['pt'], ['string
'])
>>> schema.columns
[<column num, type bigint>, <partition pt, type string>]
```

Create a Table

You can use a table schema to create a table, as shown in the following code:

```
>>> table = odps.create_table('my_new_table', schema)
>>> table = odps.create_table('my_new_table', schema, if_not_exists=
True) # Create a table only when no table exists.
>>> table = o.create_table('my_new_table', schema, lifecycle=7) # Set
the life cycle.
```

You can use a **field name field type** string connected by commas to create a table, as shown in the following code:

```
>>> # Create a non-partition table.
>>> table = o.create_table('my_new_table', 'num bigint, num2 double',
if_not_exists=True)
>>> # To create a partition table, you can input (list of table fields
, list of partition fields).
>>> table = o.create_table('my_new_table', ('num bigint, num2 double',
 'pt string'), if_not_exists=True)
```

Without related settings, you can use only the BIGINT, DOUBLE, DECIMAL, STRING, DATETIME, BOOLEAN, MAP, and ARRAY types when creating a table.

If your service is on the public cloud, or supports new data types such as TINYINT or STRUCT, you can set `options.sql.use_odps2_extension = True` to enable the new types, as shown in the following code:

```
>>> from odps import options
>>> options.sql.use_odps2_extension = True
>>> table = o.create_table('my_new_table', 'cat smallint, content
struct<title:varchar(100), body string>')
```

Obtain Table Data

Table data can be obtained using three methods:

- Call `head` to obtain table data as follows (only the first 10,000 data records or fewer of each table can be obtained):

```
>>> t = odps.get_table('dual')
>>> for record in t.head(3):
>>>     print(record[0]) # Obtain the value at the zero position.
>>>     print(record['c_double_a']) # Obtain a value through a field
.
>>>     print(record[0: 3]) # Slice action
>>>     print(record[0]) # Obtain values at multiple positions.
>>>     print(record['c_int_a', 'c_double_a']) # Obtain values
through multiple fields.
```

- Run `open_reader` on a table to open a reader to read data. You can choose to use the WITH expression:

```
# Use the with expression.
>>> with t.open_reader(partition='pt=test') as reader:
>>>     count = reader.count
>>>     for record in reader[5:10] # This action can be performed
 multiple times until a certain number (indicated by count) of
records are read. This statement can be transformed to parallel
action.
>>>         # Process a record.
>>>
>>> # Do not use the with expression.
>>> reader = t.open_reader(partition='pt=test')
>>> count = reader.count
>>> for record in reader[5:10]
>>>     # Process a record.
```

- Call the Tunnel API to read table data. The `open_reader` action is encapsulated in the Tunnel API.

Write Data

A table object can also perform the `open_writer` action to open the writer and write data, which is similar to `open_reader`. For example:

```
>>> # Use the with expression.
>>> with t.open_writer(partition='pt=test') as writer:
>>>     writer.write(records) # Here, records can be any iteratable
records and are written to block 0 by default.
>>>
>>> with t.open_writer(partition='pt=test', blocks=[0, 1]) as writer:
 # Open two blocks at the same time
>>>     writer.write(0, gen_records(block=0))
>>>     writer.write(1, gen_records(block=1)) # The two write
operations can be parallel in multiple threads. Each block is
independent.
>>>
>>> # Do not use the WITH expression.
>>> writer = t.open_writer(partition='pt=test', blocks=[0, 1])
>>> writer.write(0, gen_records(block=0))
>>> writer.write(1, gen_records(block=1))
```

```
>>> writer.close() # You must close the writer. Otherwise, the written
 data may be incomplete.
```

Similarly, writing data into the table is encapsulated in the Tunnel API. For more information, see *data upload and download channel*.

Delete a Table

Delete a table as shown in the following code:

```
>>> odps.delete_table('my_table_name', if_exists=True) # Delete a
table only when the table exists
>>> t.drop() # The drop function can be directly executed if a table
object exists.
```

Table Partitioning

- **Basic operations**

  Traverse all partitions of a table, as shown in the following code:

  ```
  >>> for partition in table.partitions:
  >>>     print(partition.name)
  >>> for partition in table.iterate_partitions(spec='pt=test'):
  >>>     Traverse list partitions.
  ```

  Check whether a partition exists, as shown in the following code:

  ```
  >>> table.exist_partition('pt=test,sub=2015')
  ```

  Obtain the partition, as shown in the following code:

  ```
  >>> partition = table.get_partition('pt=test')
  >>> print(partition.creation_time)
  2015-11-18 22:22:27
  >>> partition.size
  0
  ```

- **Create a Partition**

  ```
  >>> t.create_partition('pt=test', if_not_exists=True) # Create a
  partition only when no partition exists.
  ```

- **Delete a Partition**

  ```
  >>> t.delete_partition('pt=test', if_exists=True) # Delete a
  partition only when the partition exists.
  >>> partition.drop() # Directly drop a partition if a partition
  object exists.
  ```

SQL

PyODPS supports MaxCompute SQL query and can directly read the execution result.

- **Run the SQL statements**

```
>>> odps.execute_sql('select * from dual') # Run SQL in synchronous
mode. Blocking continues until SQL execution is completed.
>>> instance = odps.run_sql('select * from dual') # Run the SQL
statements in asynchronous mode.
>>> instance.wait_for_success() # Blocking continues until SQL
execution is completed.
```

- **Read the SQL statement execution results**

  The instance that runs the SQL statements can directly perform the `open_reader` action. One scenario is that the SQL statements return structured data, as follows:

```
>>> with odps.execute_sql('select * from dual').open_reader() as
reader:
>>>     for record in reader:
>>>         # Process each record.
```

Another scenario is that actions that may be performed by SQL, such as `desc`, obtain the raw SQL execution result through the `reader.raw` attribute, as follows:

```
>>> with odps.execute_sql('desc dual').open_reader() as reader:
>>>     print(reader.raw)
```

Resource

Resources commonly apply to UDF and MapReduce on MaxCompute.

You can use `list_resources` to list all resources and use `exist_resource` to check whether a resource exists. You can call `delete_resource` to delete resources or directly call the `drop` method for a resource object.

PyODPS mainly supports two resource types: file resources and table resources.

- **File Resources**

  File resources include the basic `file` type, and `py`, `jar`, and `archive`.

  > **Note:**
  > In DataWorks, file resources in the py format must be uploaded as files. For more information, see *Python UDF*.

  Create a File Resource

  You can create a file resource by specifying the resource name, file type, and a file-like object (or a string object), as shown in the following example:

```
resource = odps.create_resource('test_file_resource', 'file',
file_obj=open('/to/path/file')) # Use a file-like object.
```

```
resource = odps.create_resource('test_py_resource', 'py', file_obj='
import this') # Use a string.
```

Read and Modify a File Resource

You can call the `open` method for a file resource or call `open_resource` at the MaxCompute entry to open a file resource. The opened object is a file-like object. Similar to the `open` method built in Python, file resources also support the open mode. For example:

```
>>> with resource.open('r') as fp: # Open a resource in read mode.
>>>     content = fp.read() # Read all content.
>>>     fp.seek(0) # Return to the start of the resource.
>>>     lines = fp.readlines() # Read multiple lines.
>>>     fp.write('Hello World') # Error. Resources cannot be written
 in read mode.
>>>
>>> with odps.open_resource('test_file_resource', mode='r+') as fp:
 # Enable read/write mode.
>>>     fp.read()
>>>     fp.tell() # Current position
>>>     fp.seek(10)
>>>     fp.truncate() # Truncate the following content.
>>>     fp.writelines(['Hello\n', 'World\n']) # Write multiple lines
.
>>>     fp.write('Hello World')
>>>     fp.flush() # Manual call submits the update to MaxCompute.
```

The following open modes are supported:

- `r`: Read mode. The file can be opened but cannot be written.
- `w`: Write mode. The file can be written but cannot be read. Note that file content is cleared first if the file is opened in write mode.
- `a`: Append mode. Content can be added to the end of the file.
- `r+`: Read/write mode. You can read and write any content.
- `w+`: Similar to `r+`, but file content is cleared first.
- `a+`: Similar to `r+`, but content can be added to the end of the file only during writing.

In PyODPS, file resources can be opened in binary mode. For example, some compressed files must be opened in binary mode. `rb` indicates opening a file in binary read mode, and `r+b` indicates opening a file in binary read/write mode.

- **Table Resources**

Create a Table Resource

```
>>> odps.create_resource('test_table_resource', 'table', table_name
='my_table', partition='pt=test')
```

Update a Table Resource

```
>>> table_resource = odps.get_resource('test_table_resource')
>>> table_resource.update(partition='pt=test2', project_name='
my_project2')
```

DataFrame

PyODPS provides DataFrame API, which provides interfaces similar to pandas, but can fully utilize computing capability of MaxCompute. For the complete DataFrame document, see *DataFrame*.

The following is an example of DataFrame:

Note:

You must create a MaxCompute object before starting the following steps:

```
>>> o = ODPS('**your-access-id**', '**your-secret-access-key**',
             project='**your-project**', endpoint='**your-end-point
**'))
```

Here, movielens 100K is used as an example. Assume that three tables already exist, which are `pyodps_ml_100k_movies` (movie-related data), `pyodps_ml_100k_users` (user-related data), and `pyodps_ml_100k_ratings` (rating-related data).

You only need to input a Table object to create a DataFrame object. For example:

```
>>> from odps.df import DataFrame
```

```
>>> users = DataFrame(o.get_table('pyodps_ml_100k_users'))
```

View fields of DataFrame and the types of the fields through the dtypes attribute, as shown in the following code:

```
>>> users.dtypes
```

You can use the head method to obtain the first N data records for data preview. For example:

```
>>> users.head(10)
```

	user_id	age	sex	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213
5	6	42	M	executive	98101
6	7	57	M	administrator	91344
7	8	36	M	administrator	05201
8	9	29	M	student	01002
9	10	53	M	lawyer	90703

You can add a filter on the fields if you do not want to view all of them. For example:

```
>>> users[['user_id', 'age']].head(5)
```

	user_id	age
0	1	24
1	2	53
2	3	23
3	4	24
4	5	33

You can also exclude several fields. For example:

```
>>> users.exclude('zip_code', 'age').head(5)
```

	user_id	sex	occupation
0	1	M	technician
1	2	F	other
2	3	M	writer
3	4	M	technician
4	5	F	other

When excluding some fields, you may want to obtain new columns through computation. For example, add the sex_bool attribute and set it to True if sex is Male. Otherwise, set it to False. For example:

```
>>> users.select(users.exclude('zip_code', 'sex'), sex_bool=users.sex
 == 'M').head(5)
```

	user_id	age	occupation	sex_bool
0	1	24	technician	True
1	2	53	other	False
2	3	23	writer	True
3	4	24	technician	True
4	5	33	other	False

Obtain the number of persons at age of 20 to 25, as shown in the following code:

```
>>> users.age.between(20, 25).count().rename('count')
943
```

Obtain the numbers of male and female users, as shown in the following code:

```
>>> users.groupby(users.sex).count()
```

	sex	count
0	F	273
1	M	670

To divide users by job, obtain the first 10 jobs that have the largest population, and sort the jobs in the descending order of population. An example is as follows:

```
>>> df = users.groupby('occupation').agg(count=users['occupation'].
count())
>>> df.sort(df['count'], ascending=False)[:10]
```

	occupation	count
0	student	196
1	other	105
2	educator	95
3	administrator	79
4	engineer	67
5	programmer	66
6	librarian	51
7	writer	45
8	executive	32
9	scientist	31

DataFrame APIs provide the value_counts method to quickly achieve the same result. For example:

```
>>> users.occupation.value_counts()[:10]
```

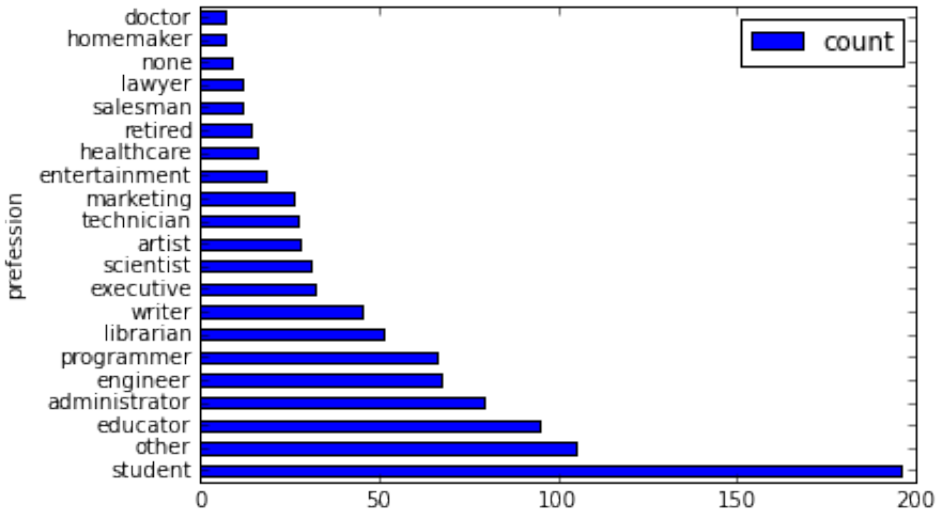
	occupation	count
0	student	196
1	other	105
2	educator	95
3	administrator	79
4	engineer	67
5	programmer	66
6	librarian	51
7	writer	45
8	executive	32
9	scientist	31

Show data in a more intuitive graph, as shown in the following code:

```
>>> %matplotlib inline
```

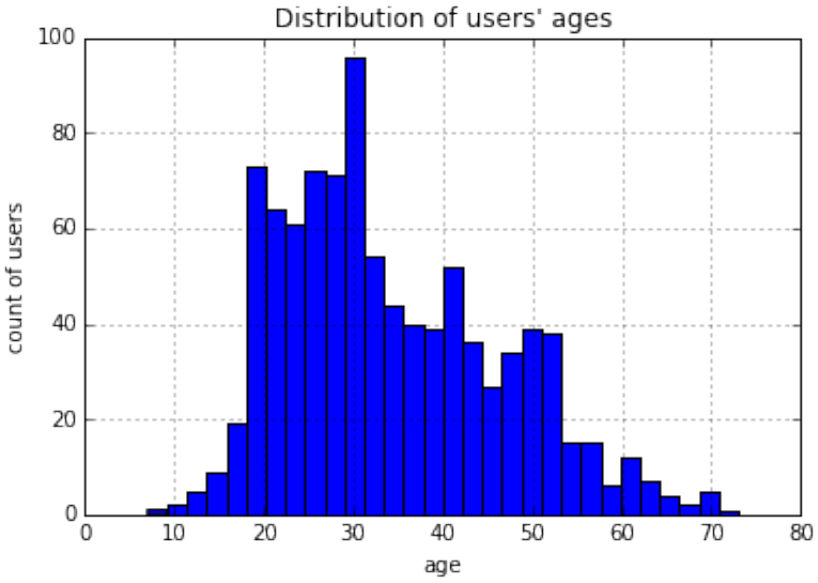
Use a horizontal bar chart to visualize data, as shown in the following code:

```
>>> users['occupation'].value_counts().plot(kind='barh', x='occupation
',
ylabel='prefession')
```



Divide ages into 30 groups and view the histogram of age distribution, as shown in the following code:

```
>>> users.age.hist(bins=30, title="Distribution of users' ages",
xlabel='age', ylabel='count of users')
```



Use JOIN to join the three tables and save the joined tables as a new table. For example:

```
>>> movies = DataFrame(o.get_table('pyodps_ml_100k_movies'))
>>> ratings = DataFrame(o.get_table('pyodps_ml_100k_ratings'))
>>> o.delete_table('pyodps_ml_100k_lens', if_exists=True)
>>> lens = movies.join(ratings).join(users).persist('pyodps_ml_
100k_lens')
>>> lens.dtypes
```

```
odps.Schema {
  movie_id int64
  title string
  release_date string
  video_release_date string
  imdb_url string
  user_id int64
  rating int64
  unix_timestamp int64
  age int64
  sex string
  occupation string
  zip_code string
}
```

Divide ages of 0 to 80 into eight groups, as shown in the following code:

```
>>> labels = ['0-9', '10-19', '20-29', '30-39', '40-49', '50-59', '60-
69', '70-79']
>>> cut_lens = lens[lens, lens.age.cut(range(0, 81, 10), right=False,
labels=labels).rename('age group')]
```

View the first 10 data records of a single age in a group, as shown in the following code:

```
>>> cut_lens['age group', 'age'].distinct()[:10]
```

	Age group	Age
0	0-9	7
1	10-19	10
2	10-19	11
3	10-19	13
4	10-19	14
5	10-19	15
6	10-19	16
7	10-19	17
8	10-19	18
9	10-19	19

View users' total rating and average rating of each age group, as shown in the following code:

```
>>> cut_lens.groupby('age group').agg(cut_lens.rating.count().rename('
total rating'), cut_lens.rating.mean().rename('average rating'))
```

	Age group	Average rating	Total rating
0	0-9	3.767442	43
1	10-19	3.486126	8181
2	20-29	3.467333	39535
3	30-39	3.554444	25696
4	40-49	3.591772	15021
5	50-59	3.635800	8704
6	60-69	3.648875	2623
7	70-79	3.649746	197

Configuration

PyODPS provides a series of configuration options, which can be obtained through `odps.options`. The following lists configurable MaxCompute options:

- **General Configurations**

Option	Description	Default value
end_point	MaxCompute Endpoint	None
default_project	Default Project	None
log_view_host	LogView host name	None
log_view_hours	LogView holding time (hours)	24
local_timezone	Used time zone. True indicates local time, and False indicates UTC. The time zone of pytz can also be used .	1
lifecycle	Life cycles of all tables	None
temp_lifecycle	Life cycles of temporary tables	1
biz_id	User ID	None
verbose	Whether to print logs	False

Option	Description	Default value
verbose_log	Log receiver	None
chunk_size	Size of write buffer	1496
retry_times	Request retry times	4
pool_connections	Number of cached connections in the connection pool	10
pool_maxsize	Maximum capacity of the connection pool	10
connect_timeout	Connection time-out	5
read_timeout	Read time-out	120
completion_size	Limit on the number of object complete listing items	10
notebook_repr_widget	Use interactive graphs	True
sql.settings	MaxCompute SQL runs global hints	None
sql.use_odps2_extension	Enable MaxCompute 2.0 language extension	False

- **Data Upload/Download Configurations**

Option	Description	Default value
tunnel.endpoint	Tunnel Endpoint	None
tunnel.use_instance_tunnel	Use Instance Tunnel to obtain the execution result	True
tunnel.limited_instance_tunnel	Limit the number of results obtained by Instance Tunnel	True
tunnel.string_as_binary	Use bytes instead of unicode in the string type	False

- **DataFrame Configurations**

Option	Description	Default value
interactive	Whether in an interactive environment	Depend on the detection value
df.analyze	Whether to enable non-MaxCompute built-in functions	True

Option	Description	Default value
df.optimize	Whether to enable DataFrame overall optimization	True
df.optimizes.pp	Whether to enable DataFrame predicate push optimization	True
df.optimizes.cp	Whether to enable DataFrame column tailoring optimization	True
df.optimizes.tunnel	Whether to enable DataFrame tunnel optimization	True
df.quote	Whether to use `` to mark fields and table names at the end of MaxCompute SQL	True
df.libraries	Third-party library (resource name) that is used for DataFrame running	None

- **PyODPS ML Configurations**

Option	Description	Default value
ml.xflow_project	Default Xflow project name	algo_public
ml.use_model_transfer	Whether to use ModelTransfer to obtain the model PMML	True
ml.model_volume	Volume name used when ModelTransfer is used	pyodps_volume

7 Handle-Unstructured-data

7.1 Access OSS Data

The following section details how to access OSS data in 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 the same account, you must log on to OSS account to authorize). Create a role through *Role Management* in the console whose name likes AliyunODPSDefaultRole 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. Authorize the role necessary permissions **AliyunODPSRolePolicy** to access OSS . For example:

```
"Version": "1",
"Statement": [

 "Action": [
   "oss:ListBuckets",
   "oss:GetObject",
   "oss:ListObjects",
   "oss:PutObject",
   "oss:DeleteObject",
   "oss:AbortMultipartUpload",
   "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 extractors. You can also use MaxCompute's internal 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*.

Create 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,
Maid 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 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://<your-access-id>:<your-secret-key>@oss-xx-xxxx-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.

  For more information about external tables, see *DDL statements*.

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.

Query 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. Examples are 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::xxxxxxxxxxxxx: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 extractors 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());
```

In the Extractor() interface, the specific read and processing logic is implemented on the audio files. The SNR of the read data is calculated based on the audio model and results are written to a record in [snr , id] format.

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 computersnr (string ID, byte [] buffer, int
validbufferlen ){
    // 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. Using a similar method, you can also use MaxCompute's large-scale computing power to easily process different types of unstructured data, such as image and video.

Data partition

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.

- Reduce access data volume: You are responsible for planning data storage address, and considering using multiple numbers of EXTERNAL TABLE to scan data in different parts, making each LOCATION of EXTERNAL TABLE point to a data subaggegate.
- 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/';
```

In contrast to the external table example provided in the previous section, in the aforementi oned three-tier partition example, when you define EXTERNAL TABLE, you specify the external table as a partition table using the syntax of PARTITION BY. The partition keys are 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 08:07:11 115 MB 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.

On this premise, partition information is imported to MaxCompute through the **`ALTER TABLE ADD PARTITION`** 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 operations are similar to the standard MaxCompute inner table operations, and you can see documentation if you are unfamiliar with the *partition tables* concept. 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). **By not performing a full scan of all** the data in the entire log_data/ **directory, a lot of useless I/O operations can be avoided**.

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.

7.2 Visit Table Store Data

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 , see *What is Table Store*.

MaxCompute and Table Store are two independent big data computing and storage services. Therefore, these two services must ensure that the network between them is open. When MaxCompute's public cloud service accesses data stored in Table Store, we recommend that you use Table Store's **private network address**, usually a host name suffixed 'ots-internal.aliyuncs.com', for example tablestore://odps-ots-dev.cn-shanghai.ots-internal.aliyuncs.com.

This document introduces how to import data from Table Store to the MaxCompute computing environment. This allows seamless connections between multiple data sources.

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

MaxCompute Type	TableStore Type
BINARY	BINARY

Authorization with STS Mode

To access Table Store data, MaxCompute requires a secure authorization channel. To address 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 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.

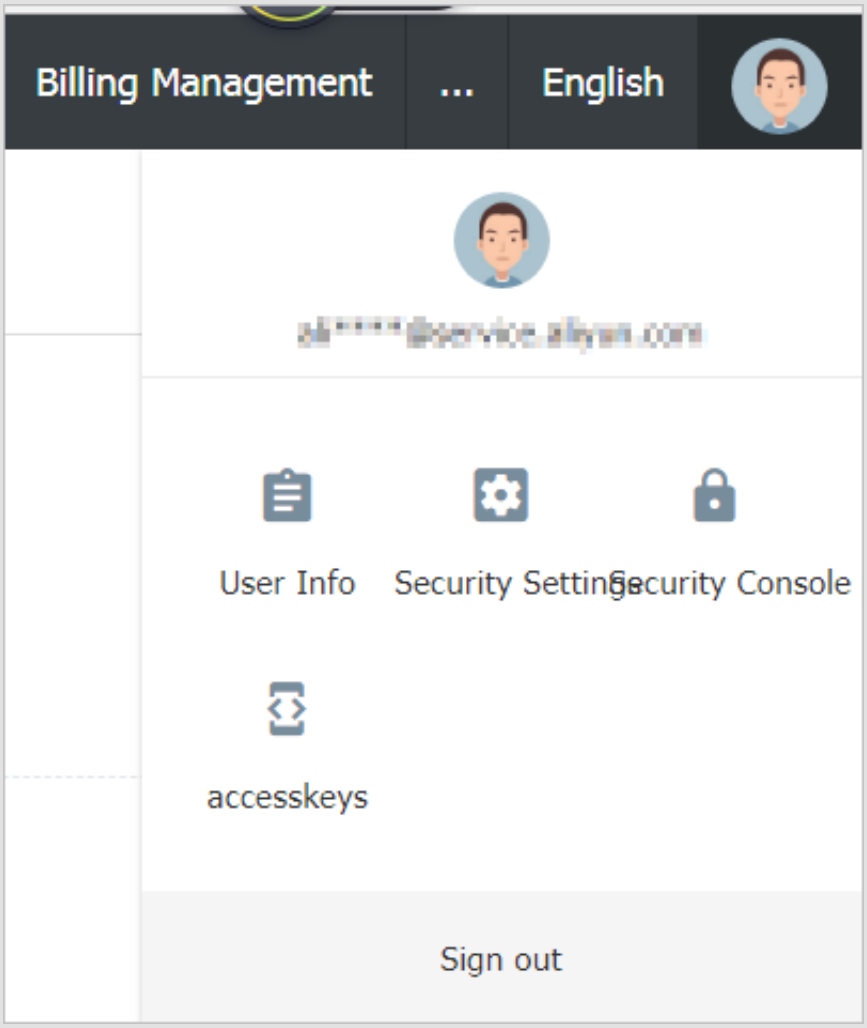
     Log on to the (if MaxCompute and Table Store are not the same account, you must log on with the Table Store account to authorize), and create the role AliyunODPSDefaultRole.
  2. Set its policy content as follows:

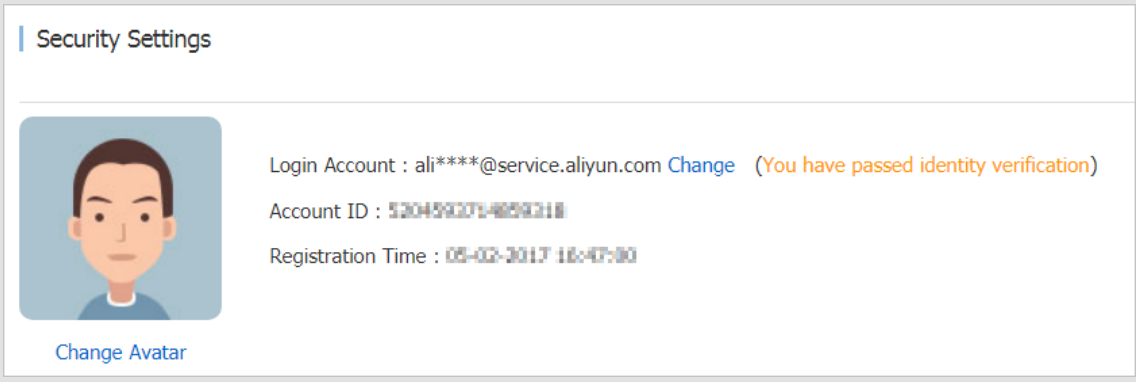
```
-- When the MaxCompute and Table Store owner are the same account
:

"Statement": [

 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "odps.aliyuncs.com"

 }


"Version": "1"

-- When the MaxCompute and Table Store owner are not the same
account:

"Statement": [

 "Action": "sts:AssumeRole",
 "Effect": "Allow",
 "Principal": {
   "Service": [
     "MaxCompute's Owner 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.

Create 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,
Maid 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', -- (3)
'tablestore.table.name'='ots_tpch_orders' -- (4)
'odps.properties.rolearn'='acs:ram::xxxxx:role/aliyunodpsdefaultrole'

LOCATION 'tablestore://odps-ots-dev.cn-shanghai.ots-internal.aliyuncs.
com'; -- (5)
```

The statement is described below:

- 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.
  - tablestore.columns.mapping option: A required parameter used to describe columns in the Table Store tables accessed by MaxCompute, including primary key and attribute columns.
    - At the beginning of the column name, `:` ndicates 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.
  - tablestore.table.name: The name of the Table Store table to access. 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.
  - The information in odps.properties.rolearn comes from the Arn information of AliyunODPSDefaultRole in RAM. You can get it through the **details of the role** in the RAM console.
- The LOCATION clause is used to specify specific Table Store information, including the AccessKey of OSS owner, the instance name and endpoint. Because you must specify the AccessKey the of OSS owner, to avoid disclosing the AccessKey of your primary account, we recommend that you use RAM user credentials.

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 and column names when creating the external table.

If you perform **multiple computations** on a single data set, instead of remotely reading data from Table Store each time, you can import all the necessary data to MaxCompute, to create a MaxCompute (internal) table. For example:

```
CREATE TABLE internal_orders AS
SELECT odps_orderkey, odps_orderdate, odps_custkey, odps_totalprice
FROM ots_table_external
WHERE odps_orderkey > 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;
```

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. For more details, see *MaxCompute access to OTS data*.

8 Graph

8.1 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. Vertices and edges contain values.

MaxCompute Graph supports the following graph editing operations:

- Editing the value of Vertex or Edge.
- Adding/deleting Vertex.
- Adding/deleting Edge.

Note:

When editing a vertex and an edge, you must maintain their relationship.

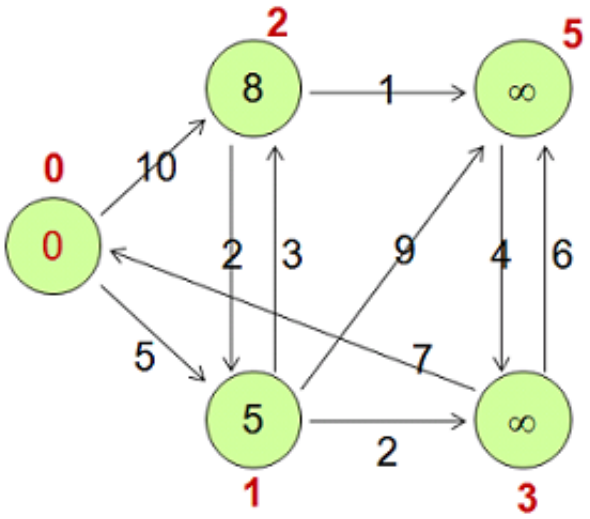
This process outputs a final solution after performing iterative graph editing and evolution. Typical applications include *PageRank*, *SSSP algorithm*, and *Kmeans algorithm*. You can use Java SDK, an interface provided by MaxCompute Graph, to compile graph computing programs.

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 indicate the vertex ID (ID), value (Value), status (Halted, indicating whether an iteration is to be stopped), and edge set (Edges, indicating lists of all edges starting from the vertex). The edge structure can be described as < DestVertexID, Value >, which respectively indicate the destination vertex (DestVertexID) and value (Value).



For example, the preceding figure consists of the following vertices.

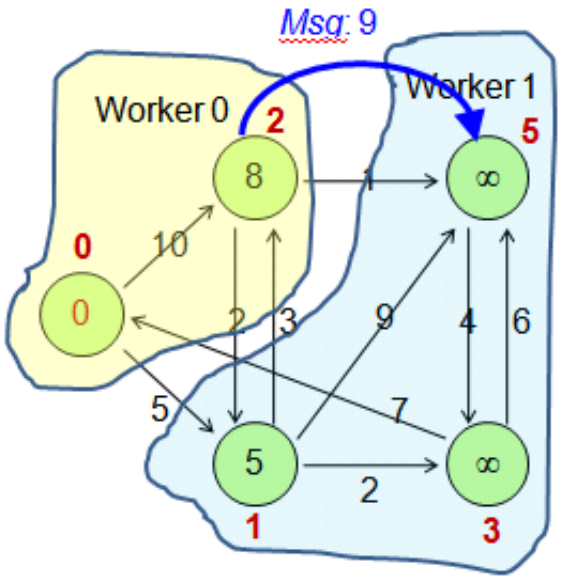
Vertex	<ID, Value, Halted, Edges>
v0	<0, 0, false, [<1, 5 >, <2, 10 >] >
v1	<1, 5, false, [<2, 3>, <3, 2>, <5, 9>]>
v2	<2, 8, false, [<1, 2>, <5, 1 >]>
v3	<3, Long.MAX_VALUE, false, [<0, 7>, <5, 6>]>
v5	<5, Long.MAX_VALUE, false, [<3, 4 >]>

Graph program logic

Graph loading

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: Calculate the hash value of a vertex ID and perform 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 because the result of the ID mod 2 is 1.

Iteration calculation

- An iteration is called a **superstep**. It traverses all vertices in the non-halted status (the value of Halted is false) or all vertices that receive messages (a vertex in halted status is automatically woken up after receiving a message), and calls their compute (ComputeContext context, Iterable messages) method.
- You can follow these steps on your implemented compute (ComputeContext context, Iterable messages) method:
  - Process Messages sent from previous SuperStep to 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 global information.
  - Set the current vertex to the 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 (only if any of the following conditions is met)

If any of the following conditions is met, iteration becomes terminate.

- All vertices are in the halted status (the value of Halted is true) and no new message is generated.
- The maximum number of iterations is reached.
- The terminate() method of an Aggregator returns true.

The pseudocode is described 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();
```

8.2 Function overview

Running job

The MaxCompute console provides JAR commands to run MaxCompute Graph jobs. These command are used in the same way as to run *MapReduce JAR commands*.

This document introduces 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 references the value of the Graph job and runs in a distributed environment, while -classpath references 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, you can run the main function of the Graph job to directly submit the job to MaxCompute, rather than submitting the 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

MaxCompute Graph jobs must be input and output using tables.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 GraphJob. The framework reads records in the input table and transmits them to custom GraphLoader to load data.
- Restrictions: Currently, shard filtering conditions are not supported. For more information about restrictions on applications, see *Application restrictions*.

The following example shows how to define a job output. Multiple job outputs are supported. Each output is marked using 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 restrictions on applications, see *Application restrictions*.

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 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 about restrictions on applications, see *Application restrictions*.

8.3 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 ID, value, halted, and edges attributes. A vertex is implemented by the setVertexClass interface of GraphJob.
Edge	Edge is the abstract of edges in a graph, including the attributes destVertexId and value. Adjacency 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 for graph topology modification. 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 calculation 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 custom logic during startup and exit. WorkerComputer is implemented by using the setWorkerComputerClass interface of GraphJob.

Main interface	Description
Aggregator	setAggregatorClass(Class …) defines one or multiple Aggregators.
Combiner	setCombinerClass sets a Combiner.
Counters	Indicates the counter. In job running logic, the WorkerContext interface can be used to obtain Counters and perform counting. The framework automatically sums results.
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.

8.4 Development and Debugging

MaxCompute does not provide Graph development plug-ins for users. However, you can develop the MaxCompute Graph program based on Eclipse. The development process is as follows:

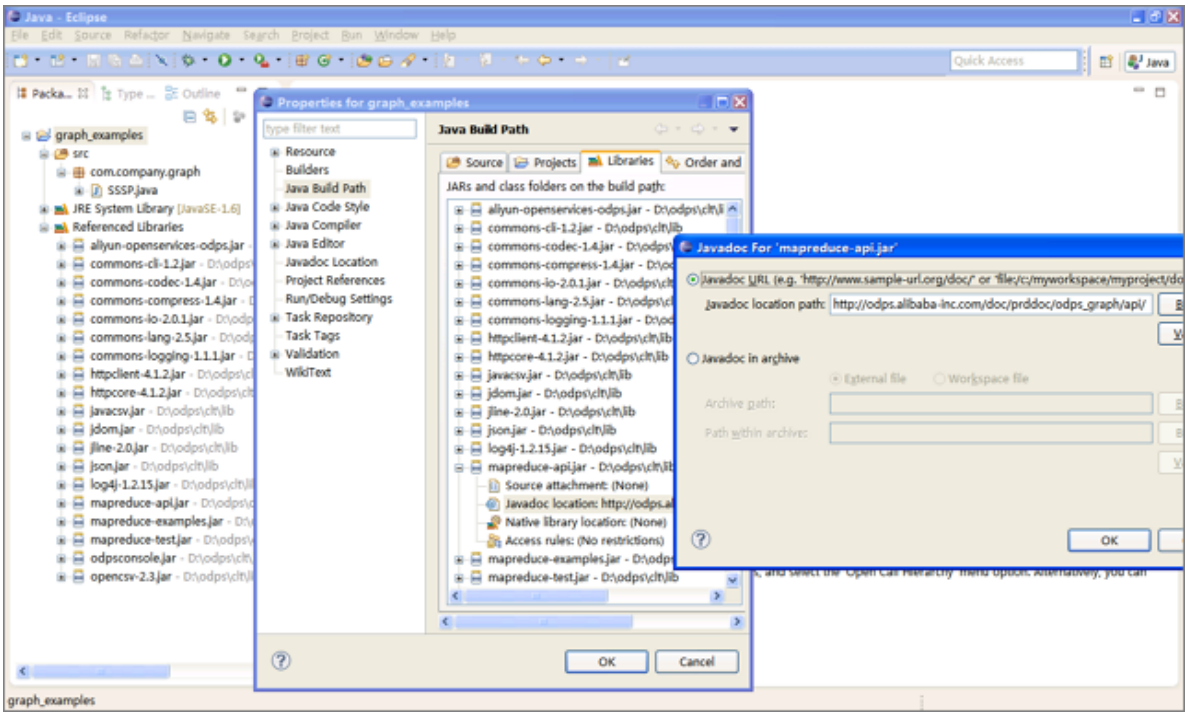
1. Compile Graph codes and perform basic tests using local debugging.
2. Perform cluster debugging and verify the result.

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 the 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

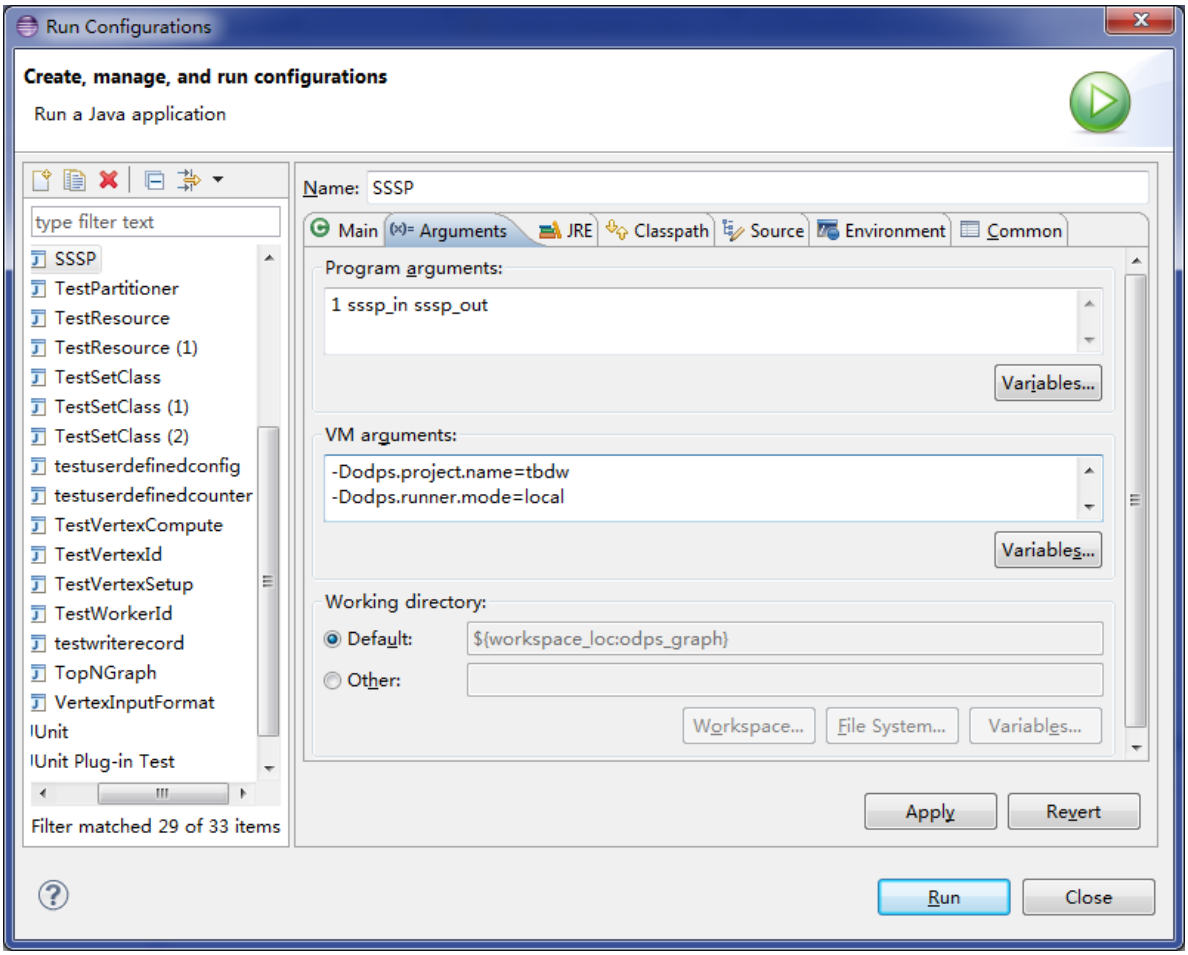
MaxCompute Graph supports the local debugging mode. You can use Eclipse to perform breakpoint debugging.

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: Specifies the current project, which is required.
> - odps.end.point: Specifies the address of the current MaxCompute service, which is optional. 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 when 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.

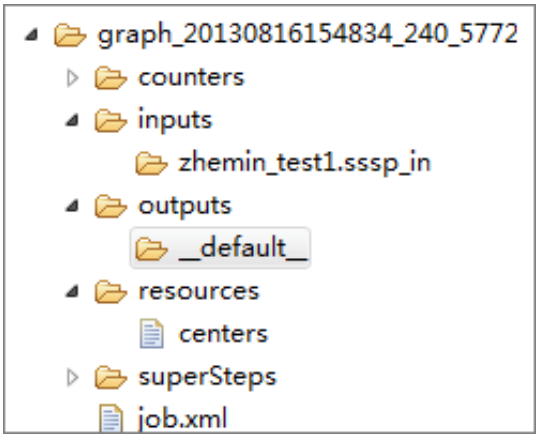
After SSSP debugging is implemented locally in Eclipse, the following information is output:

```
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, result data in outputs overwrites 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, you can submit the job to a cluster for testing.

The procedure is as follows:

1. Configure the MaxCompute client.
2. Run the `add jar /path/work.jar -f;` command to update the JAR package.
3. Run the 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 in 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 for faster processing.

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 long running 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

You can define a Combiner to reduce the memory that stores messages and network data traffic volume, shortening 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, you must check whether an implementation method exists in commons-codec-1.3.jar or wait for MaxCompute to upgrade to a supported version.

8.5 Restriction

The restrictions 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. (The number of input tables cannot exceed 64). The number of outputs of one job cannot exceed 256.
- Label conventions are as follows: A label can be up to 256 characters in length and can contain letters, numbers, 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].
- The number of times for repeatedly reading a resource by one Worker cannot exceed 64.
- The split size is 64 MB by default, which can be set. 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*.

8.6 Examples

8.6.1 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.

8.6.2 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 recalculat es 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.

8.6.3 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.

8.6.4 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();
```

8.6.5 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");
```

8.6.6 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");
```

8.6.7 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");
```

8.6.8 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");
```

8.6.9 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");
```

8.6.10 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();
```

8.6.11 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();
```

8.7 Introductions of Aggregator Mechanism

This document describes the implementation and related APIs of Aggregator and uses KmeansClustering as an example to illustrate the usage of Aggregator.

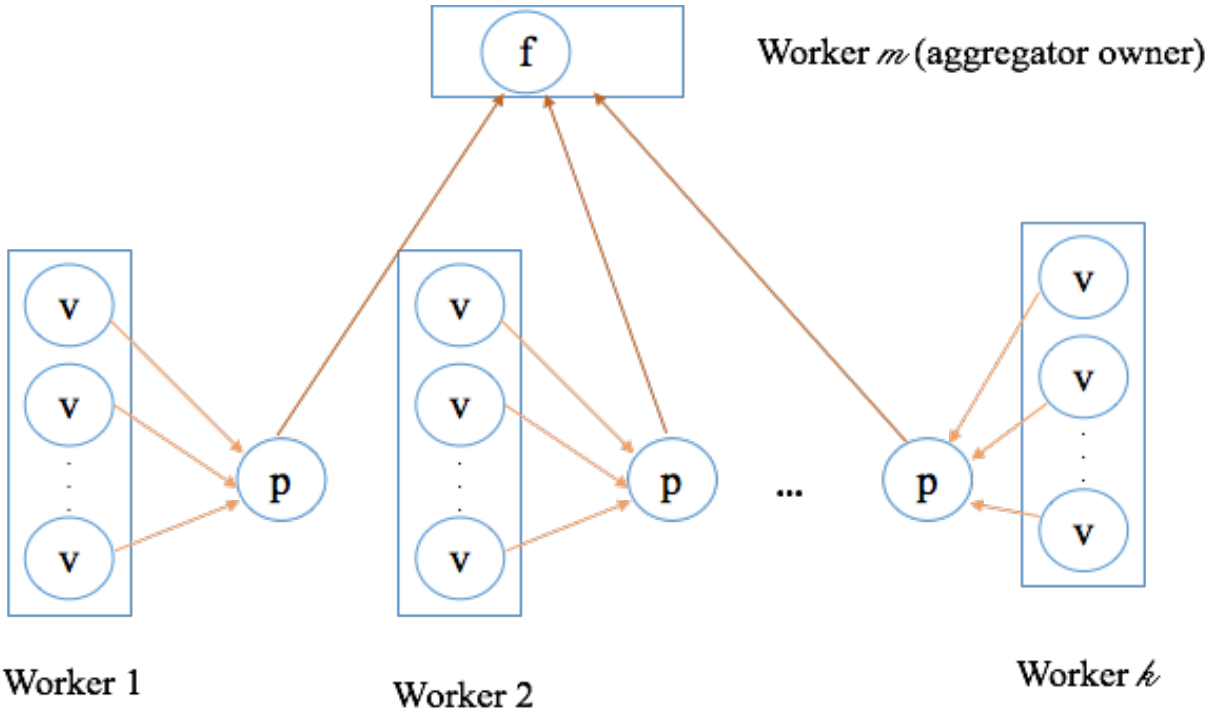
In MaxCompute Graph, Aggregator helps collect and process global information. In MaxCompute Graph, Aggregator is used to summarize and process global information.

Aggregator Implementation

The logic of Aggregator is divided into two parts.

- One part is run on all Workers in distributed mode,
- and the other part is only run on the Worker where AggregatorOwner is located in 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 where AggregatorOwner is located aggregates partial aggregation objects sent by common Workers to obtain a global aggregation result, and determines whether the iteration ends. 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 is run once on all Workers. It is called before all supersteps start, and is generally used to initialize AggregatorValue. 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 starts.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 is run 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 run, 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 is run 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 aggregatio n results of p0, p1, and p2. If p1, p0, and p2 in sequence are sent to the Worker where AggregatorOwner is located in, the merge sequence is as follows:

  1. merge(p1, p0) is run first, and p1 and p0 are aggregated as p1'.
  2. merge(p1', p2) is run, and p1' and p2 are aggregated as p1'', which is the global aggregatio n result in this superstep.

  The preceding example shows that execution of the merge() method 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 is continued. 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 Aggregator.

 **Note:**

The complete code is provided in the 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.

  First, 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 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**

The custom KmeansVertex code is as follows. Regarding its logic, Partial aggregation is run 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, recording the number of samples closest to each center. sums and counts are used together to calculate a new center, which is a main content of aggregation.

The next is KmeansAggregator used for custom Aggregator implementation. The following describes implementation in order of the preceding APIs.

1. Run createStartupValue().

```
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().

```
@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 attachment.)

The preceding three functions are run on all Workers to implement partial aggregation.

The following describes global aggregation-related operations 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 that values of sums and counts aggregated by each Worker are added together.

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 Workers is not partitioned based on Partitioner. Data is maintained by the Worker that loads the data.The data loaded is no longer repartitioning Based on the partition, that is, who loads the data and who maintains it.

Conclusion

This document introduces the aggregator mechanism in the MaxCompute graph, the API meaning, and the kmeans Clustering example. In general, Aggregator can be implemented as follows:

1. Each Worker runs createStartupValue during startup to create AggregatorValue.

2. Each Worker runs createInitialValue before each iteration starts to initialize 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 Security

9.1 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.

9.2 Quick Start

9.2.1 Add users and grant permissions

In the following scenario, Jack is the project administrator of a project called prj1. A new team member named Alice, who already has the Alibaba Cloud account alice@aliyun.com), applies to join the prj1project. Alice requests the following permissions: view table lists, submit jobs, and create tables.

The following procedure is performed by Jack, the project administrator:

```
    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
```

9.2.2 Add users and grant permissions using ACL

In the following scenario, Jack is the project administrator of a project called prj1. In the scenario, three new data auditors, Alice, Bob, and Charlie, are added to the project team. They need to all apply for the following permissions: view table lists, submit jobs, and read the table userprofile.

In this scenario, the project administrator can perform authorization by using the object-based *ACL Authorization* .

The following procedure is performed by Jack, the project administrator:

```
    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;
```

9.2.3 Project data protection

In the following scenario, Jack is the project administrator of a project called prj1. The project involves a large volume of sensitive data including user IDs, shopping records, and data mining algorithms with proprietary intellectual property rights. Jack wants to properly protect the sensitive data and algorithms and restrict project users to only have access to data within the project. Jack wants to make sure that data can only flow within the project. Many data mining algorithms with proprietary intellectual property right are included in the project. Jack hopes that these sensitive data and algorithms can be properly protected and only accessible to users within the project, and that data flow in the project only, and not out of the project.

To protect the project data, Jack must follow 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 data flows only within the project.

There may be scenarios where users want to export data tables out of the project under the approval of the project administrator. In such cases, MaxCompute provides the TrustedProject configuration to support external data export from the protected project. In this type of scenario, to configure project prj2 as a trusted project of prj1 and to enable data flow from prj1 to prj2, perform the following command:

```
    use prj1;
    add trustedproject prj2;
```

9.3 User Authentication

MaxCompute supports the **Alibaba Cloud account system** and the **RAM account system** .

 **Note:**

MaxCompute can recognize the RAM account system but cannot recognize the RAM permission system. Users can add any of their RAM sub-accounts to a project of MaxCompute. 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, you can 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:
You need a valid email address when you apply for an Alibaba Cloud account. The email address is then 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 is 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 from disclosure. When an AccessKey needs updated, you can create a new AccessKey and disable the existing AccessKey.

Log on to MaxCompute with an Alibaba Cloud Account

You must configure the AccessKey in the configuration file conf/odps_config.ini before you use odpscmd to log on. The following is an 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, it takes 15 minutes after the operation is completed.

9.4 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. If the OS is Linux, run **`./bin/odpscmd`**. If the OS is Windows, run **`./bin/odpscmd.bat`**.

Add a User

In the following scenario, 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 on the official page, 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 following:

- 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 do not need 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 permission to perform operations related to `account providers`.

As shown in the preceding figure, you can only see the `ALIYUN` account system. If you want to add RAM accounts support, you can run the `add accountprovider ram;` as follows:

```
odps@ odps_pd_inter>add accountprovider ram;
OK
```

After added successfully, the user still cannot operate MaxCompute. The user must be granted certain permissions to operate MaxCompute within the limits of permissions. For more information, see *Authorization*.

User Authorization

After the user has been 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 owner of the project 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 owner of the project 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`* and *`bob@aliyun.com`* RAM sub-account, can logon to MaxCompute with their own **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 Create a *RAM user*.
- For more information about how to add or remove users on MaxCompute, see the corresponding 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 a user is 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 any authority to access resources of the project.
- Before removing a user who has been assigned roles, those roles must be revoked. 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 the permission data from a project.

Alice To remove corresponding users, Alice can run the following two 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. sub-account 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
```

9.5 Role Management

A role is a defined set of access permissions. It can be used to assign the same set of permission s to a group of users. Role-based authorization can greatly simplify the authorization process and reduce the authorization management cost. Role-based authorization can be used with priority when user authorization is performed. Role-based authorization can greatly simplify the authorizat ion procedure and reduce authorization management costs. When a user must be authorized, the owner should consider whether it would be better to use a role to authorize them.

When a project is created, an admin role is automatically created with a defined set of privileges authorized to the role. These privileges include access to all objects within the project, management, and authorization of users and roles. In comparison to a project 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 commands are as follows:

```
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 also belong to 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

Use the following command format to create a role:

```
CREATE ROLE;
```

For example,

assume the role player must be created. Enter the following command on the client:

```
create role player;
```

Add a User to Role

To add a user to the role, use the following command format to add a user to the role:

```
GRANT <roleName> TO <full_username> ;
```

For example,

assume the user bob@aliyun.com must be added to 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.

For example,

suppose 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 procedure is as follows:

```
    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;
```

Revoke the Role from a User

To revoke the role from a user, use the following command format to revoke the role from a user:

```
REVOKE <roleName> FROM <full_username>;
```

For example,

assume the user bob@aliyun.com must be removed from the player role. Enter the following command on the client:

```
revoke player from bob@aliyun.com;
```

Delete a Role

To delete a role, use the following command format to delete a role:

```
DROP ROLE <roleName>;
```

For example,

assume the role player must be deleted:

```
drop role player;
```

Note:
When you delete 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. If there are such users, this role cannot be removed. Removing a role succeeds only if all of its users are already revoked from it.

9.6 Authorization

After a *user* is added, the project owner or the project administrator must authorize the user. The user can perform operations only after obtaining permission. Authorization allows a user to perform operations including read, write, and view on tables, tasks, resources, and other objects of the MaxCompute.

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.
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 the access control list (ACL) is considered as a type of sub-resource of the object. Authorization can be performed only when 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 the GRANT and REVOKE commands defined by SQL92 or with Oracle database security management can find 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 must 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 following authorization operations procedure is performed by the project administrator:

```
    use test_project; --Open the project
    add user aliyun$alice@aliyun.com; --Add the user
    add user aliyun$alice@aliyun.com; --Add the user
    create role worker; --Create a role
    grant worker TO aliyun$alice@aliyun.com; --Grant the role
    grant worker TO aliyun$bob@aliyun.com; --Grant the role
    grant CreateInstance, CreateResource, CreateFunction, CreateTabl
e, List ON PROJECT test_project TO ROLE worker; --Authorize the role
```

- **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 following authorization operations procedure is performed by the administrator 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, only Resource authorization is required. Write as the following:

```
create function function_name as 'com.aliyun.odps.compiler.udf.
PlaybackJsonShrinkUdf' using 'test_project_b/resources/odps-compiler-
playback.jar' -f;.
```

9.7 Permission Check

MaxCompute provide 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
```

View the 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.
```

For 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;
```

View RAM sub-account permissions:

```
show grants for RAM$account:sub-account;
```

For example:

```
show grants for RAM$bob@aliyun.com:Alice;
```

View the Permissions of a Specified Role

```
describe role --View access permissions granted to a specified role
```

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 omitted, the default type is Table.

9.8 Security Configuration

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. Note: 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*.

9.9 Security Command List

9.9.1 Security Configuration of a Project

Authentication Configuration

Statements	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.

9.9.2 Permission Management of a Project

User Management

Statement	Description
list users	View all users added to the project.
add user <username> `<username>`	Add a user.
remove user <username> `<username>`	Remove a user.

Role Management

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 a user.
revoke `<rolelist>` from `<username>`	Revoke the role from a 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 permissions and role.

Statement	Description
show acl for `<objectName>` [on type `<objectType>`]	View specific object authorization information.
describe role `<roleName>`	View role authorization information and role assignments.

9.10 Resource share across project space

9.10.1 Resource Sharing across Projects Based on Package

Assume that you are the owner or administrator (admin role) of some projects. One of your primary accounts has multiple projects, wherein the prj1 project 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 of 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 need to be controlled by a user in fine-grained manner, and the user who applies for the control permission is a member of the business project team, we recommend that you use 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.

Without Package the following problem cannot be effectively solved.

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 on them: 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 authorizat ion 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.

9.11 Column-level Access Control

Label-based security (LabelSecurity) is a required 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.

Differences 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.

A user who wants to read data in a MaxCompute project must first apply for the SELECT permission, similar to the person who wants to drive in a country must first apply for a driver's license. The permission application is within the scope of DAC.

Because the country has a high traffic accident rate, it adds a statute against drunk driving. All drivers are required to have a driver's license and consume no alcohol before driving. In MaxCompute, prohibition against reading of highly sensitive data is analogous to the statute 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 arising from the management of data sensitivity levels by 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 beyond 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 overwrites the sensitivity label set for the table, without consideration for 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:

After the label-authorized permission of a user to access a table is revoked, this user's permission to access the table fields 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 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**

  Scenario 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. If a user needs to access the sensitive data for business purposes, the user must be authorized by the project owner or administrator.

Alice is a member of the project. For business 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 granting. This issue can be avoided if the project administrator sets the column sensitivity properly.

- **Prohibit the Project Users with Access to Sensitive Data from Copying and Distributing the Sensitive Data within the Project without Authorization**

  In the preceding use case, Alice is granted the access permission on the data with a sensitivity level of 2 for business 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 prohibit such behavior on Alice.

  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.
```