Alibaba

ODPS User Manual

Preface

LEGAL INFORMATION

Contents

Tables

1 Overview

This document faces to ODPS users, mainly to introduce some common operations, detailed SQL statements, functions supported by ODPS, MapReduce instances, installation and usage for tools, etc.

- Basic operations: in this part, some basic operation commands used in ODPS will be listed, including project operations, table operations, job operations, instance operations, resource operations, function operations, etc.
- ODPS SQL: ODPS SQL provides data query and processing functions for the external interface. In this part, some acceptable operators and language (including DDL, DML, built-in function, UDF) will be introduced.
- ODPS Mapreduce: ODPS provides MapReduce programming interface. User can use Java API provided by MapReduce to write MapReduce program for processing data in ODPS. In this part, the data processing flows of MapReduce, Mapreduce functions, some program examples will be introduced.
- Tools: in this part, the installation and usage of tools used in ODPS will be introduced, including ODPS client (CLT) and Eclipse plug-in.

For basic introduction and some concepts of ODPS, please refer to ***ODPS Quick Start Manual.***

2 Basic Operations

2.1 Project Operations

2.1.1 Enter Project

Command Format:

```
use <project_name>;
```

Action:

Enter the specified project. After entering this 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, return exception.

Example:

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

🕮 **Note:**

- The example given above is running in ODPS client. All ODPS command keywords, project names, table names, column names are case insensitive.

After running this command, user can access the objects of this project. Suppose the table test_src has existed in the project ‘my_project’ and run the following command:

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

ODPS will search the tables under my_project automatically. If the table exists, return the data of this table. If the table does not exist, exception is thrown. To access the table test_src in another project ‘my_project2’ through the project ‘my_project’, specify the project name as follows:

```
odps:my_project>sql; --Enter the sql shell in client. SQL can be executedonly in this shell
odps:sql:my_project>select * from my_project2.test_src;
```

Now return the data in my_project2 instead of the data of test_src in my_project.

ODPS does not provide the commands to create and delete the project. User can complete more configurations and operations through ODPS Console.

2.1.2 Query All Projects

Command format:

```
odps@ > list projects;
```

Use: query all projects in ODPS.

2.1.3 Other Operations

1) Recyclebin management-- view objects in recyclebin

Command Format:

```
odps@ > show recyclebin;
```

Use: list object details of this project

2) Recyclebin management-- purge objects in recyclebin

Command Format:

```
odps@> purge all;
```

Use: purge all objects in the recyclebin of this project to release storage space. Only the project owner can do this operation.

3) Purge the objects with specified table name in recyclebin.

Command Format

```
odps@> purge table tblname;
```

Use: Purge the objects with specified table name in recyclebin to release storage space. If the table with specified name exists, project owner or the user who has ‘write’ privilege can execute this operation. If the table has been dropped, only the project owner can execute this operation.

2.2 Table Operations

2.2.1 Create Table

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.

Example:

```
CREATE TABLE IF NOT EXISTS sale_detail(
 shop_name      STRING,
 customer_id    STRING,
 total_price    DOUBLE)
PARTITIONED BY (sale_date STRING,region STRING);  --If there is no same name table existing, create a partition table.
```

📖 **Note:**

- The table name and column name are both case insensitive.
- The table name and column name cannot have special characters. It can only begin with a letter and include a-z, A-Z, digits and underline_. The name length cannot exceed 128 bytes.
- The comment content is the effective string which length can not exceed 1024 bytes.

2.2.2 Drop Table

Command Format:

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

Action:

Delete a table.

If the option [if exists] is not specified and the table does not exist, then return exception. If this option is specified, no matter whether the table exists or not, all return success.

Description:

Table_name: the table name to be deleted.

Example:

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

2.2.3 Describe Table

Command Format:

```
DESC <table_name>;
```

Action:

Return the information of specified table, including Owner, Project, CreateTime, LastDDLTime, LastModifiedTime, InternalTable (it indicates the object to be described is table and always shows YES), Size (storage size occupied by table data, unit is Byte), Native Columns (no-partition column information, incuding column name, type, comment), Partition Columns (partition column information, including partition name, type, comment).

Parameter

table_name: table name or view name

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 example shown above is executed in ODPS client.
- If the table has no partition, the information of Partition Columns will not be displayed.
- To describe a view, the option 'InternalTable' will not be displayed but the option 'VirtualView' will be displayed and its value is always YES. Similarly, the option 'Size' will be replaced by ViewText. For example: select * from src.

2.2.4 Show Tables

Command Format

```
SHOW TABLES;
```

Action:

List all tables of current project.

Example:

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

📖 **Note:**

- The example shown above is to be executed in ODPS client.
- ALIYUN is system prompt, indicating the ALIYUN user.
- odps_user@aliyun.com is user name, indicating the creator of the table.
- Table_name refers to table name.

2.2.5 Show Partitions

Command Format:

```
SHOW PARTITIONS <table_name>;
```

Action:

List all partitions of a table.

Parameter:

table_name: Specify the table to be queried. If the table does not exist or it is not a partition table, the exception will be thrown.

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 example shown above is to be executed in ODPS client.
- Partition_col1 and partition_col2 indicate the partition columns.
- col1_value1, col2_value1, col1_value2, and col2_value2 indicate corresponding partition values.

2.3 Job Operations

2.3.1 Submit Job by Command

Command Format:

```
run job <jobname> [cron(expression)] [Parameters(name:value [,])]
```

Use: Start a new subjob according to the parameter, corn expression and job command.

2.3.2 Show Job List

Command Format:

```
list jobs;
```

Use: Display all defined subjobs in current project.

2.3.3 Show Job

Command Format:

```
show job <jobname> [from 2012-11-01] [to 2012-11-01] ;
```

Use: display all running instances corresponding to the job.

2.4 Instance Operations

2.4.1 Show Instances

Command Format:

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

Action:

Return the instances created by current users.

Parameters:

Startdate, enddate: return the instances during specified time period (from startdate to enddate). Support the following format: yyyy-mm-dd, precision to the day. It is optional parameter; return the instances submitted by users in three days if not specified.

Number: specify the number of instance to be returned. In accordance with the time scheduling, return N (number) instances nearest the current time. If not specified, return all instances that meet the requirement.

The output items include: StartTime (the time accurate to seconds), RunTime(s), Status (including Waiting, Success, Failed, Running, Canceled, and Suspended), InstanceID and corresponding SQL:

StartTime	RunTime	Status	InstanceID	Request
2013-12-25 17:35:13	7s	Success	2013122509351320g2hojk4y2	select null is null as c from src;
...	...	...	...	...
...	...	...	...	...

There are six kinds of Task states:

- Running
- Success
- Waiting
- Failed
- Suspended
- Cancelled

2.4.2 Status Instance

Command Format:

```
STATUS <instance_id>;
```

Action:

Return the status of specified instance, which includes: Success, Failed, Running, and Canceled.

If this instance is not created by current user, expection will be thrown.

Parameter Description

instance_id: the unique identifier of an instance, to specify which instance to be queried.

Example:

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

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

2.4.3 Kill Instance

Command Format:

```
kill <instance_id>;
```

Action:

Stop specified instance and the status of it should be ‘Running’. It is worth to mention that this is an abnormal process and it does not mean that the distributed task has stopped after the system accepts the request and returns result. So whether the instance is killed should be confirmed after using ‘status’ command to view the instance status after running this command.

Parameters:

instance_id: the unique identifier of an instance, which should be ID of an instance which status is ‘Running’, otherwise, an error will be thrown.

Example:

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

Stop the instance which ID is 20131225123302267gk3u6k4y2.

2.5 Resource Operations

2.5.1 Create Resource

Command Format:

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

Parameter Description:

File/archive/table/py indicates the resource type.

local_file: path of local file, and take this file name as the resource name. Resource name is a unique identifier of a resource.

table_name: table name in ODPS.

[PARTITION (spec)]: when the resource to be added is a partition table, ODPS only supports taking a partition as a resource, not a whole partition table.

Alias: specify the resource name. If not added this parameter, the file name is default to be the resource name. Jar and Py resources do not support the fuction.

[comment 'cmt']: add comment for resource.

[-f]: if the same name resource exists, to add this parameter will cover the original resource. If this parameter is not added and the same name resource exists, the operation will fail.

📖 **Note:**

- At present, each resource file size can not exceed 64MB. The resource size referenced by Single SQL or MapReduce can not exceed 512MB.

2.5.2 Delete Resource

Command fomat:

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

Parameter Description:

resource_name: a specified resource name.

2.5.3 List Resources

Command fomat:

```
LIST RESOURCES;
```

Description:

Check all resources of current project.

Example:

```
odps@ $project_name>list resources;

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

2.6 Function Operations

2.6.1 Function Register

Command Format:

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

Description:

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). For python UDF, this name is python script name.class name. This parameter must be in quotes.

resource_list: resources list used by UDF. The resource which contains UDF code must be incuded in the list. If the user's code reads the resource file by 'distributed cache' interface, this list will also contain the resource file list UDF reads. The resource list is composed of multiple resource names, separated by comma (,). The resource list must be in quotes.

Example: Suppose that the Java UDF class org.alidata.odps.udf.examples.Lower is in my_lower.jar. Create the UDF 'test_lower':

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

Suppose that Python UDF MyLower is in the script 'pyudf_test.py', create the function 'my_lower':

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

Note:

- Like the resource file, the same name UDF can only be registered once.
- Generally UDF can not 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 information will be printed in Summary after SQL execution.

2.6.2 Remove Function

Command Format:

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

Example:

```
DROP FUNCTION test_lower;
```

2.7 ALIAS Command

The use of Alias is to read different resources (data) through a fixed resource name in ODPS M/R or UDF codes without modifying code.

Command Format:

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

Use: create alias for resource.

Example:

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

The resource alias above is 'resName' and different resource tables are referenced in two jobs. Different data can be read without modifying the code.

2.8 Other Operations

2.8.1 Log

All jobs of ODPS are running in a distributed environment. By using Log command, user can view the status of each process (Worker) in a job.

Command Format:

```
LOG <action> [OPTIONS]
```

Action includes: get, help and list. The function of each action can be viewed through 'help action'. For example, view the function of 'list action':

```
odps@ test_project>log help ls;
usage: log ls
 -d,--duration          sort by duration
 -F,--failed            select failed entries
 -i <instance id>       is required if not specified once
 -l,--limit <arg>       maximum log entry number
 -r,--reverse           reverse the sort order of results
 -R,--running           select running entries
    --raw               get detail's raw json
 -t <task name>
```

📖 **Note:**

- The option '-d' is to sort by the length of each worker execution time.
- The option '-r' refers to reverse sort, which must be used in conjunction with

'-d'.

Next give an example to use log command:

1) Suppose that the table exists and run following SQL:

```
SELECT CAST(sale_date AS DATETIME) FROM sale_detail;
  -- This is an ODPS SQL, to run it and an error will be thrown. Here user does not have to
care about the failure reason.
```

2) Get the following output:

```
odps@ test_project>select cast(sale_date as datetime) from sale_detail;

ID = 20140116065603291go4musna
...
FAILED: ODPS-0121095:Invalid arguments - in function cast, string datetime's format
must be yyyy-mm-dd hh:mi:ss,   input string is:201310,
 column hint is 'sale_detail.sale_date'
```

3) Check each worker status of this SQL:

```
odps@ odps_test_smode>log ls -i 20140116065603291go4musna;
M1_Stg1#0_0           1970-01-01 08:00:00        0s            Failed
```

If you do not understand SQL distributed implementation, ignore the significance of the first item 'M1_Stg1#0_0'. But please note that this field can uniquely identify each worker in SQL execution time. The second column represents the start time of each worker. The third item indicates work time of each worker. (Due to the data in the example is very few, which is only in two rows, the execution time is less than 1 second.) The last column 'Failed' indicates the worker status.

4) View stdout and stderr of the process (worker):

```
odps@ odps_test_smode>log get M1_Stg1#0_0 -type stderr;
agrv 0   = cgcreate
agrv 1   = -a
agrv 2   = admin
agrv 3   = -g
agrv 4   = memory

odps@ odps_test_smode>log get M1_Stg1#0_0 -type stdout;
LimitKey:core
SetLimit - core:unlimited
[2014-01-16 14:56:45.456484] -----Run Task: M1_Stg1#0 ----------
```

In this example, the helpful error information cannot be obtained from 'stdout' and 'stderr'. But at some point, stdout and stderr information can help users to investigate the problem. The user who has distributed experience is proposed to use this function.

📖 **Note:**

- The function of log command is very powerful; the most basic usage scenario is mentioned above. Users can obtain more detailed description through 'log help'.

2.8.2 Set

Command Format:

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

Description: User can use set command to set system variable of ODPS or the user defined variable.

At present, the system variables supported in ODPS include:

```
--Set commans supported by ODPS SQL and Mapreduce (new version):

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 ODPS specified job.
 The priority is lower than above three 'set key'. Unit is M and no default value.

set odps.stage.mapper.split.size=      --Modify the input data quantity of each map
worker; that is the size of input file burst. -- Thus control the worker number of each map
stage. Unit is M and the default value is 256M

 set odps.stage.reducer.num=      --Modify the worker number of each reduce stage and
 no default value.

 set odps.stage.joiner.num=      - Modify the worker number of each join stage and no
 default value.

 set odps.stage.num=             --Modify the worker concurrency of all stages in ODPS
 specified job. The priority is lower than above three 'set key' and no default value.


 --Set commands supported by ODPS MapReduce (old version)

set odps.mapred.map.memory=      --Set the memory size of each map worker. Unit is M
```

```
 and default value is 1024M.
set odps.mapred.reduce.memory= Set the memory size of each reduce worker. Unit is M
and default value is 1024M.


set odps.mapred.map.split.size=
--Modify the input data quantity of each map worker; that is the size of input file burst.
-- Thus control the worker number of each map stage. Unit is M and the default value is
256M
 set odps.mapred.reduce.tasks=   --Modify the worker number of each reduce stage and
 has no default value.
```

2.8.3 SetProject

Command Command:

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

Description:

User can use ‘setproject’ command to set project attribute.

If the value of <KEY>=<VALUE> is not specified, the current project attribute configuration will be displayed.

The detailed description of project attributes is shown as follows:

Table 1-1 Project Attributes

Attribute Name	Who sets privilege	Attribute Description	Value Range
odps.table.drop.ignorenonexistent	All users	Whether an error will be thrown if deleting a table which does not exist.	True (no error reported)/false
odps.instance.priority.autoadjust	ProjectOwner	Whether to adjust task priority automatically and let small task schedule in high priority.	True /false
odps.instance.priority.level	ProjectOwner	Adjust the priority of tasks in Project.	1 (highest priority) ~ 3
odps.security.ip.whitelist	ProjectOwner	Specify IP whitelist to access project.	IP list, separated bycomma.

Attribute Name	Who sets privilege	Attribute Description	Value Range
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. Mandatory: the lifecycle clause is mandatory. 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	Remain days of instance information	3 ~ 30
odps.task.sql.outerjoin.ppd	ProjectOwner	In full outer join statement, whether the filtration condition can be pushed down.	true/false
odps.function.strictmode	ProjectOwner	When the built-in function meets the dirty data, return NULL (true) or throw exception (false).	true/false
odps.task.sql.write.str2null	ProjectOwner	Whether to take empty string as NULL (true)	true/false

2.8.4 Export Project Meta

Command Format:

```
export <projectname> <local_path>;
```

Use: Export peoject Meta to a local file. Meta shows as the statement accepted by odpscmd. User can rebuild a project by using this ‘meta’ file.

2.8.5 Show Flags

Display the parameters set by ‘Set command’.

Command Format:

```
show flags;
```

Description:

Running ‘Use Project’ command will clear the configuratioin set by ‘Set’ command.

3 ODPS SQL

3.1 Summary

3.1.1 Application Scenarios

ODPS SQL is suitable for the scenarios: there is massive data (TB level) to be processed and real-time requirement is not high. It takes seconds or even minutes to prepare each job and submit each job, so ODPS SQL is not acceptable for the services which need to process thousands to tens of thousands of transactions per second.

The synax of ODPS SQL is similar to SQL. It can be considered as a subset of standard SQL. But ODPS 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. At present, the maximum length of SQL in ODPS is 2MB.

3.1.2 Reserved Word

ODPS SQL considers the keywords of SQL statement as reserved words. It colud not be used to nominate table, column or partition; otherwise an error will be thrown. Reserved words are case insensitive. The reserved words in common use are shown as follows:

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

3.1.3 Type Conversion

3.1.3.1 Explicit Type Conversion

Explicit type conversion is a behavior to convert the value of one data type to the value of another type by 'cast'. The explicit type conversions to be supported by ODPS SQL are shown as Table 1-2:

Table 1-2 Explicit Type Conversion Rule

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

'Y': can be converted; 'N': can not be converted. '－': no need to be converted

Note:

- To convert the type 'double' to 'bigint', the fractional part will be truncated. For example: cast (1.6 as bigint) = 1.
- To convert the type 'string' to 'bigint' and the string meets 'double' format, the string will be converted to 'double' at first and then converted to 'bigint'. So the decimal part will be truncated. For example: cast ("1.6" as bigint) = 1.
- The type 'string' which meets 'bigint' format can be converted to 'double' and keep one decimal place. For example: cast ("1" as double) = 1.0.
- The explicit type conversion not to be supported will result in abnormality.
- If the conversion fails during execution, report error and exit.
- To convert the date type, use the default format yyyy-mm-dd hh: mi: ss.
- Some types can not be converted by explicit type conversion, but can be converted through SQL built-in functions. For example: to convert the type 'boolean' to 'string', the function 'to_char' can be used. While the function 'to_date' also supports the conversion from 'string' to 'datetime'.
- For the cast introduction, please refer to Other Functions .

3.1.3.2 Implicit Type Conversion

Implicit type conversion is an automatic type conversion by ODPS according to context use environment and type conversion rules. The implicit type conversion supported by ODPS is the same as explicit type conversion:

Table 1-3 Implicit Type Conversion Rule

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

'Y': can be converted; 'N': can not be converted. '－': no need to be converted

Note:

- The implicit type conversion not to be supported will cause abnormality.
- If the conversion fails during execution process, it will also cause abnormality.
- Since the implicit type conversion is an automatic conversion according to the context envirement, therefore, 'cast' explicit conversion is recommended if the types do not match the rule.
- The implicit type conversion rule is effective in some scopes. For details, please refer to scope of implicit type conversion.

3.1.3.2.1 Relational Operators (=, <>, <, <=, >, >=, IS NULL, IS NOT NULL)

The relational operators include: =, <>, <, <=, >, >=, IS NULL, IS NOT NULL, LIKE, RLIKE and IN. Since the implicit conversion rule of LIKE, RLIKE and IN is different from other relational operators, the description of these three operators will be introduced in next section. The description in this section does not contain these three special operators. When different types of data are involved in relational operation, follow the rules below to do implicit type conversion:

Table 1-4 Implicit Type Conversion Rule with Retional Operator

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

Datetime	N	N	Datetime	－	N
Boolean	N	N	N	N	－

Note:

- If the two types can not do implicit type conversion, then the relational operation will not be completed and report error to exit.
- For the introduction of relational operators, please refer to 3.2.1.

3.1.3.2.2 Special Relational Operators (LIKE, RLIKE, IN)

The usage of LIKE and RLIKE shown as follows:

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

Notes of these two items in implicit type conversion:

- The parameters ‘source’ and ‘pattern’ of LIKE and RLIKE can only be strings.
- Other type to be involved in operation is not allowed and the implicit type conversion to ‘string’ is not supported.

The usage IN as follows:

```
key in (value1, value2, …)
```

The implicit conversion rules of IN:

- The data types in value list on the right side of IN should be consistent.
- To compare key and values, if bigint, double and string are compared, convert them to double in uniformity. If the datetime and string are compared, convert them to datetime in uniformity. Except these, the conversion between other types is not allowed.

3.1.3.2.3 Arithmetic Operators

The Arithmetic Operators include: +, -, *, /, %, +, -. Its implicit conversion rule is shown as follows:

Only the types ‘string’, ‘bigint’ and ‘double’ can be involved in computation. Before computation, ‘string’ will be converted to ‘double’ by implicit type conversion. When ‘Bigint’ and ‘double’ are involved in arithmetic operation, ‘bigint’ will be converted to ‘double’. The types ‘datetime’ and ‘Boolean’ are not allowed in arithmetic operations.

3.1.3.2.4 Logic Operators

The logic operators include: ‘and’, ‘or’ and ‘not’ and corresponding implicit conversion rule are listed as follows:

- Only the type ‘boolean’ can be involved in logic operation.
- Other types are not allowed involving in logic operation so the implicit type conversion of them is also not allowed.

3.1.3.2.5 ODPS SQL Built-in Function

ODPS SQL provides many system functions, which are convenient for user to calculate one or more columns of any row and output any kinds of data type. Its implicit conversion rules are listed as follows:

- To invoke a function, if the data type of input parameter is not consistent with the data type defined in function, convert the data type of input parameter to the function defined data type.
- Parameters of each ODPS SQL built-in function have different requirements for allowed implicit type conversion.

3.1.3.2.6 CASE WHEN

The implicit conversion rules of Case When:

- If return types are bigint and double, uniform them to double.
- If there is a ‘string’ type in rentun types, uniform all types to ‘string’. If it cannot be converted, an error will be reported (such as Boolean).
- Except these, the conversion of other types is not allowed.

3.1.3.2.7 Partition Column

ODPS SQL supports partition table. For the definition of partition table, please refer to the descriptions in 3.3.1DDL and 3.3.2 DML. At present, ODPS partition just supports string type and the conversion of any other types is not allowed.

3.1.3.2.8 UNION ALL

To be involved in UNION ALL operation, the data type of column, column number and column name must be consistent, otherwise expection will be thrown.

3.1.3.2.9 Conversion between String and Datetime

ODPS supports the conversion between string and datetime. The format to be used in conversion is yyyy-mm-dd hh: mi: ss.

Table 1-5 Conversion between String and Datetime

Unit	String (ignore case)	Effective Value Range
Year	yyyy	0001 ~ 9999
Month	mm	01 ~ 12
Day	dd	01 ~ 28,29,30,31
Hour	hh	00 ~ 23
Minute	mi	00 ~ 59
Second	ss	00 ~ 59

Note:

- In the value range of each unit, if the first digit is 0, it can not be omitted. For example, '2014-1-9 12:12:12' is an illegal datetime format and it cannot be converted from 'string' to 'datetime' and must be written as '2014-01-09 12:12:12'.
- Only the string type which meets the format description mentioned above can be converted to datetime type. For example, cast ("2013-12-31 02:34:34" as datetime) will convert the string '2013-12-31 02:34:34" to datetime. Similarly, when converting datetime to string, the default format to be converted is 'yyyy-mm-dd hh: mi: ss' format. For example, the following conversions will cause exception:

  cast ("2013/12/31 02/34/34" as datetime)

  cast ("20131231023434" as datetime)

  cast ("2013-12-31 2:34:34" as datetime)

It is worth to note that the threshold of 'dd' part depends on actual days of a month. If the value exceeds the actual days, it will cause abnormally exiting. For example:

```
 cast("2013-02-29 12:12:12" as datetime)        --return exception. 2013-02-29 did not
exist.
 cast("2013-11-31 12:12:12" as datetime)        --return exception. 2013-11-31 did not
exist.
```

ODPS provides the function 'to_date', used to convert 'string' type which does not meet datetime format to 'datetime' type. For details, please refer to TO_DATE..

3.2 Operator

3.2.1 Relational Operators

Operator	Description
A=B	If expression A or expression B is NULL, return NULL. TRUE if expression A is equal to expression B otherwise FALSE.
A<>B	If expression A or expression B is NULL, return NULL. TRUE if expression A is not equal to expression B otherwise FALSE.
A<B	If expression A or expression B is NULL, return NULL. TRUE if expression A is less than expression B otherwise FALSE.
A<=B	If expression A or expression B is NULL, return NULL. TRUE if expression A is less than or equal to expression B otherwise FALSE.
A>B	If expression A or expression B is NULL, return NULL. TRUE if expression A is greater than expression B otherwise FALSE.
A>=B	If expression A or expression B is NULL, return NULL. TRUE if expression A is greater than or equal to expression B otherwise FALSE.

A IS NULL	If expression A is NULL, return TRUE. Otherwise, return FALSE.
A IS NOT NULL	If expression A is not NULL, return TRUE. Otherwise, return FALSE.
A LIKE B	If expression A or expression B is NULL, return NULL. TRUE if string A matches the SQL simple regular expression B, otherwise FALSE. The comparison is done character by character. The _ character in B matches any character in A, and the % character in B matches an arbitrary number of characters in A (similar to .* in posix regular expressions). 'aaa' like 'a__' = TRUE 'aaa' like 'a%' = TRUE 'aaa' like 'aab' = FALSE 'a%b' like 'a\%b' = TRUE 'axb' like 'a\%b' = FALSE
A RLIKE B	NULL if expression A or B is NULL, TRUE if any substring of A matches the Java regular expression B, otherwise FALSE. If expression B is empty, report error and exit.
A IN B	B is a set. If expression A is NULL, return NULL. If expression A is in expression B, return TRUE, otherwise return FALSE. If expression B has only one element 'NULL', that is, A IN (NULL), return NULL. If expression B contains NULL element, take NULL as the type of other elements in B set. B must be a constant and at least has one element; all types should be consistent.

3.2.2 Arithmetic Operators

Operator	Description
A + B	If expression A or B is NULL, return NULL; otherwise return the result of adding A and B. Expression A and B should be bigint.
A – B	If expression A or B is NULL, return NULL; otherwise return the result of subtracting B from A. Expression A and B should be bigint.
A * B	If expression A or B is NULL, return NULL; otherwise return the result of multiplying A and B. Expression A and B should be bigint.
A / B	If expression A or B is NULL, return NULL; otherwise return the result of dividing B from A. Expression A and B should be bigint and the result is double.
A % B	If expression A or B is NULL, return NULL; otherwise return the reminder resulting from dividing A by B.
+A	Still return A.
-A	If expression A is NULL, return NULL; otherwise return –A.

Note:

- Only string, bigint and double could be involved in arithmetic operation, date type and Boolean type are not allowed participating in arithmetic operation. Before involved in operation, the type String will be converted to double by implicit type conversion. If bigint and double are both involved in arithmetic operation, the type bigint will be converted to double by implicit type conversion.

3.2.3 Bitwise Operators

Operator	Description
A & B	Return the result of bitwise AND of A and B. For example: 1&2, return 0; 1&3, return 1; Bitwise AND of NULL and other values, all return NULL. Expression A and B should be bigint.
A | B	Return the result of bitwise OR of A and B. For example: 1 |2, return3. 1 |3, return 3. Bitwise OR of NULL and other values, all return NULL. Expression A and B should be Bigint.

🕮 **Note:**

- Bitwise operator does not support implicit conversion, only supports the type Bigint.

3.2.4 Logical Operators

Operator	Description
A and B	TRUE and TRUE=TRUE TRUE and FALSE=FALSE FALSE and TRUE=FALSE NULL and FALSE=FALSE TRUE and NULL=NULL NULL and TRUE=NULL NULL and NULL=NULL
A or B	TRUE if either A or B or both are TRUE, otherwise FALSE. TRUE or TRUE=TRUE TRUE or FALSE=TRUE FALSE or TRUE=TRUE FALSE or NULL=NULL NULL or FALSE=NULL TRUE or NULL=TRUE NULL or TRUE=TRUE NULL or NULL=NULL
NOT A	TRUE if A is FALSE, otherwise FALSE. Return NULL if A is NULL.

🕮 **Note:**

- Only the type Boolean can be involved in logic operation and the implicit type conversion is not supported.

3.3 Language

3.3.1 DDL (Data Definition Language)

3.3.1.1 Table operations

3.3.1.1.1 CREATE TABLE

Statement 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
```

Description

- The table name and column name are both case insensitive.
- Exception will be thrown if a same name table has already existed.User should specify the option [if not exists] to skip the error. If the option [if not exists] is specified, no matter whether there is a same name table, even if the source table structure and the target table structure are inconsistent, all return success. The Meta information of existing table will not change.
- The data types will only support: bigint, double, Boolean, datetime and string.
- The table name and column name cannot have special characters. It can only begin with a letter and include a-z, A-Z, digits and underline_. The name length cannot exceed 128 bytes.
- Use ‘Partitioned by’ to specify the partition and at present only string and bigint are supported. The partition value can not have a double byte characters (such as Chinese), must begin with a letter a-z or A-Z, followed by letter or numbe. The name length cannot exceed 128 bytes. Allowed characters including: space ‘ ’, colon ‘:’, underlined symbol ‘_’, ’$’, ’#’, point ‘.’, exclamation point ‘!’ and' ‘@’. Other characters are taken as undefined characters, such as ‘\t’, ‘\n’, ‘/’ and so on. If using partition fields in partition table, a full table scan is no need when adding partition, updating data in partition and reading data in partition, to improve the processing efficiency.
- The comment content is the effective string which length does not exceed 1024 bytes.

- Lifecycle indicates the lifecycle of the table. The statement 'create table like' will not copy the lifecycle attribute from source table.
- At present, the partition hierarchy cannot exceed 5 levels.

Next example is used to create a table sale_detail to save sales records. Use sale date (sale_date) and sale region (region) as the partition columns:

```
CREATE TABLE IF NOT EXISTS sale_detail(
 shop_name        STRING,
 customer_id     STRING,
 total_price     DOUBLE)
PARTITIONED BY (sale_date STRING, region STRING); --If there is no same name table existing, create a partition table.
```

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

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

If the table sale_detail has data, the example metioned above will copy all data of sale_detail into the table sale_detail_ctas1. Please note that sale_detail is a partition table, while the table created by the statement 'create table ... as select ...' will not copy the partition attribute. The partition column of source table will become a general column of object table. That is to say, sale_detail_ctas1 is a non-partition table with 5 columns.

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

```
create table sale_detail_ctas2 as
      select shop_name,
            customer_id,
            total_price,
            '2013' as sale_date,
            'China' as region
from sale_detail;
```

If not adding the alias of column, such as:

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

Then the forth column and fifth column of created table sale_detail_ctas3 will become system generated names, like '_c3', '_c4'. To use the table sale_detail_ctas3 again, you need to add reverse quotes so that it can be correctly cited, such as:

```
select `_c3`, `_c4` from sale_detail_ctas3;
```

To execute the SQL 'select c3, _c4 from sale_detail_ctas3' directly, an error will be reported, because the column name of ODPS SQL is not allowed beginning with space and it must be added with reverse quote. Using alias is suggested to avoid the situation metioned above.

If the destination table should have the same structure like source table, try to use 'create table ... like' statement, such as:

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

Now the table structure of sale_detail_like is exactly the same as sale_detail. Except the lifecycle priority, the column name, column comment and table comment are all the same. But the data in sale_detail can not be copied into the table sale_detail_like.

3.3.1.1.2 DROP TABLE

Statement Format:

```
drop table [if exists] table_name;
```

Description:

- If the option [if exists] is not specified but the table does not exist, then return exception. If this option is specified, no matter whether the table exists or not, all return success.

Example:

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

3.3.1.1.3 RENAME TABLE

Statement Format:

```
alter table table_name rename to new_table_name;
```

Description:

- Rename operation is just to update the table name, but not to modify the data in table.
- If the table which has the same name with 'new_table_name' has already existed, report error.
- If the table 'table_name' does not exist, report error.

Example:

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

3.3.1.1.4 Alter Table Comment

Command Format:

```
alter table table_name set comment 'tbl comment';
```

Description:

The table table_name must have existed. The acceptable max comment length is 1024 bytes.

Example:

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

Use the command ‘desc’ to view the comment modification in the tale.

3.3.1.1.5 Modify Lifecycle of Table

ODPS 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;
```

Description

- 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 ODPS will automatically recycle the table without user intervention (similar to ‘drop table’ operation). In ODPS, once the data in table is modified, the LastDataModifiedTime will be updated. So ODPS will judge whether to recycle this table based on the setting of LastDataModifiedTime and lifecycle. Suppose that the table ‘table_name’ is a partition table. ODPS will judge whether to recycle the table according to LastDataModifiedTime of each partition.

Differently from no-partition table, after the last partition of a partition table has been recycled, the table will not be 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.
```

3.3.1.1.6 Alter LastDataModifiedTime

ODPS SQL supports ‘touch’ operation to modify LastDataModifiedTime of a table.

The result is to modify ‘LastDataModifiedTime’ of a table to be current time.

Statement Format:

```
alter table table_name touch;
```

Description:

- If the table ‘table_name’ does not exist, return error.
- This operation will change the value of ‘LastDataModifiedTime’ of a table and now ODPS will consider the table data has changed and corresponding lifecycle calculation will begin again.

3.3.1.2 View Operations

3.3.1.2.1 CREATE VIEW

Statement Format:

```
create [or replace] view [if not exists] view_name
[(col_name [comment col_comment], ...)]
[comment view_comment]
[as select_statement]
```

Description:

- 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 need to 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’ will cause 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;
```

3.3.1.2.2 DROP VIEW

Statement Format:

```
drop view [if exists] view_name;
```

Description:

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

Example:

```
drop view if exists sale_detail_view;
```

3.3.1.2.3 RENAME VIEW

Statement Format:

```
alter view view_name rename to new_view_name;
```

Description:

- If the same name view has already existed, an error will be reported.

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;
```

3.3.1.3 Column and Partition Operation

3.3.1.3.1 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 ...)
```

Description:

- If the same name partition has already existed and the option [if not exists] is not specified, return exception.
- At present, the maximum number of partitions supported by ODPS is 40000.
- For multi-partition table, 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.
```

3.3.1.3.2 DROP PARTITION

Statement Format:

```
alter table table_name drop [if exists] partition_spec;
   partition_spec:
: (partition_col1 = partition_col_value1, partition_col2 = partiton_col_value2 ...)
```

Description:

- If the partition does not exist and the option [if exists] is not specified, then an error will be thrown.

Example: delete a partition from the table sale_detail.

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

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

3.3.1.3.3 Add Column

Statement Format:

```
alter table table_name add columns (col_name1 type1, col_name2 type2...)
```

Description:

- The data type of column can only be bigint, double, Boolean, datetime and string.

3.3.1.3.4 Modify Column Name

Statement Format:

```
alter table table_name change column old_col_name rename to new_col_name;
```

Description:

- ‘old_col_name’ must be an existent column.
- The column named ‘new_col_name’ cannot exist in the table.

3.3.1.3.5 Modify Column (partition) Comment

Statement Format:

```
alter table table_name change column col_name comment 'comment';
```

Description:

- The length of comments can not exceed 1024 bytes.

3.3.1.3.6 Modify LastDataModifiedTime of Table (Partition)

ODPS 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', ...);
```

Description:

- If 'table_name' or 'partition_col' does not exist, return error.
- If the specified partition_col_value does not exist, return error.
- This operation will change the value of 'LastDataModifiedTime' in a table and now ODPS will consider the data of table or partition has changed and the lifecycle calculation will begin again.

3.3.2 DML (Data Manipulation Language)

3.3.2.1 Insert

INSERT OVERWRITE/INTO

Statement Format:

```
insert overwrite|into table tablename [partition (partcol1=val1, partcol2=val2 ...)]
select_statement
from from_statement;
```

📖 **Note:**

- Insert syntax of ODPS is different from MySQL or Oracle Insert syntax. The keyword 'table' should be added following 'insert overwrite|into'. The table name cannot be added following it directly.

'Insert overwrite/into' is used to save calculation results into a destination table. The difference between 'insert into' and 'insert overwrite' is that 'insert into' will insert added data into the table or partition, while 'insert overwrite' will clear source data from the table or partition before inserting data in it. In the course of processing data through ODPS SQL, 'insert overwrite/into' is the most common statement, which can save the calculation result into a table, provided for next calculation. For example, use following statements to calculate the sale detail of different regions from the table sale_detail:

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

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

It is worth to note that the correspondence between source table and destination table depends on the column sequence in 'select' clause, nor the column name correspondence between two tables. The following statement is still legal:

```
insert overwrite table sale_detail_insert partition (sale_date='2013', region='china')
    select customer_id, shop_name, total_price from sale_detail;
---After creating the table sale_detail_insert, the column sequence is: shop_name string,
customer_id string, total_price bigint.
--While inserting data from sale_detail to sale_detail_insert, the sequence is customer_id,
shop_name, total_price.
--Now the data of sale_detail.customer_id will be inserted into
sale_detail_insert.shop_name. The data of sale_detail.shop_name will be inserted into
sale_detail_insert.customer_id.
```

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

```
insert overwrite table sale_detail_insert partition (sale_date='2013', region='china')
    select shop_name, customer_id, total_price, sale_date, region from sale_detail;
--Report error. Sale_date and region are partition columns, cannot appear in Insert
statement of static partition.
```

MULTI INSERT

ODPS 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
    [insert overwrite | into table tablename2 [partition ...]
        select_statement2]
```

Description:

- Generally, up to 128 ways of output can be written in a single SQL statement. Once exceeding 128 ways of output, report sytax error.
- In a 'multi insert' statement, for a partition table, a target partition can not appear for multiple times; for a no-partition table, this table can not appear for multiple times.
- Different partitions within a partition table cannot have both an 'insert overwrite' operation and an 'insert into' operation, otherwise, report an error.

As follows:

```
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
   insert overwrite table sale_detail_multi partition (sale_date='2011', region='china' )
      select shop_name, customer_id, total_price;
--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,
```

Output to 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;
```

Description:

- At present, a single worker can only output up to 512 dynamic partitions in a distributed environment, otherwise it will lead to abnormality.
- Currently, any dynamic partition SQL can not generate more than 2,000 dynamic partitions; otherwise it will cause abnormality.
- The value of dynamic partition can not be NULL, otherwise expection will be thrown.
- If the destination table has multiple partitions, it is allowed to specify parts of partitions to be static partitions through 'Insert' statement, but the static

partitions must be advanced partitions.

Next is a simple example to explain dynamic partition:

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

As mentioned above, user is unable to know which partitions will be 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;
insert overwrite table sale_detail_dypart partition (sale_date, region)
    select * from sale_detail;
--Return successfully.

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.

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.

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

3.3.2.2 Select

SELECT Operation

Statement Format:

```
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]
```

Description:

- 'Select' operation can read columns from table by specifying the column name. Or use '*' to read all columns. A simple Select statement is shown as follows:

```
select * from sale_detail;
```

Or just read one colume 'shop_name' in sale_detail:

```
select shop_name from sale_detail;
```

In 'where' clause, user can specify the filter condition, such as:

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

🕮 **Note:**

- If you need screen display after executing Select statement, only 1000 rows of results can be displayed. If 'select' is a clause, the 'select' clause will return all results to upper query without this restriction.

- The filter conditions supported by 'where' clause include:

Filter Condition	Description
> , < , =, >=, <=, <>	
like, rlike	
in, not in	If adding subquery behind of 'in/not in', only return one column result and the quantity of return value can not exceed 1000.

You can specify partition range in 'Where' clauses of 'Select' statement to avoid a full table scan. As follows:

```
select sale_detail.*
   from sale_detail
   where sale_detail.sale_date >= '2008' and sale_detail.sale_date <= '2014';
```

The clause 'Where' of ODPS does not support querying through 'between' condition.

- Nested query in table_reference is supported, such as:

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

- Distinct: If there are repeated rows, using 'distinct' in front of the field will remove the duplicated value and only return one value, while using 'all' will return all repeated values. If 'distinct' is not specified, the default result is the same as 'all'. Use 'distinct' to return a row of record, as follows:

```
select distinct region from sale_detail;
select distinct region, sale_date from sale_detail;
-- Multiple columns ‘distinct’; the scope of ‘distinct’ is a column set of ‘select’ statement, not a single column.
```

- Group by: a grouping query caluse, usually used with aggregation functions. When ‘Select’ statement contains aggregation functions, next items should be noted:

    1. The key of ‘group by’ can be the column name of input table.
    2. It can also be an expression consisting of the input table columns, but can not be the output column alias of ‘select’ statement.
    3. The priority of rule 1 is high than rule 2. If rules 1 and rule 2 have conflict, that is to say, the ‘group by’ key is the column or expression of input table also is the output column of ‘Select’ statement, follow rule 1.

If like:

```
select region from sale_detail group by region;
-- It‘s OK to use input table column directly as the ‘group by’ key.
select sum(total_price) from sale_detail group by region;
--it’s OK to group by region, return total price of each group.
select region, sum(total_price) from sale_detail group by region;
--It’ OK to group by region, return the region value of each group (unique in group) and total price.
select region as r from sale_detail group by r;
--Use the alias of select column and report error.
select 'China-' + region as r from sale_detail group by 'China-' + region;
--User must use the whole expression of column.
select region, total_price from sale_detail group by region;
--Report error. All columns of ‘select’ statement, which do not contain aggretion function,
 must appear in ‘group by’ clause.
select region, total_price from sale_detail group by region, total_price;
--It’s OK.
```

These restrictions depend on that ‘group by’ operation precedes ‘select operation’ in SQL analysis. So the ‘group by’ key can only be input table column or expression.

- Order by: Do globe sorting in accordance with a few columns for all data. If user wants to sort records in descending order, he can use DESC as keyword. Because it is a global sorting, ‘order by’ must be used together with ‘limit’. To sort through ‘order by’ clause, NULL would be considered smaller than any value. This behavior is consistent with Mysql, but not in conformity with Oracle. Differently from ‘group by’, ‘order by’ must be followed with the column alias of Select statement. While selecting a column, if the alias is not specified, the column name will be lised as the column alias.

```
select * from sale_detail order by region;
```

```
--Report error. '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;
--Report error. 'Order by' must be followed by column alias.
select region as r from sale_detail order by r;
```

- The number in [limit number] is a constant, to limit the output rows quantity. If the 'limit' option is not specified and using 'select' statement to view results from screen directly, only output 5000 rows mostly. This screen display limit of each project may be different and user can control it via console panel.
- Distribute by: do hash slice for data according to the values of several columns.The output column alias of 'Select' must be used.

```
select region from sale_detail distribute by region;
--It' OK because the column name is alias.
select region as r from sale_detail distribute by region;
--Report error. The end must be added the column alias.
select region as r from sale_detail distribute by r;
```

- Sort by: for partial ordering, 'distribute by' must be added in fornt of it. In fact, 'sort by' is to do partial ordering for the result of 'distribute by'. The output column alias of Select statement must be used.

```
select region from sale_detail distribute by region sort by region;
select region as r from sale_detail sort by region;
--no 'distribute by', report error.
```

- 'Order by' can not be used together with 'distribute by/sort by' clause. At the same time, 'group by' is also not allowed using together with 'distribute by/sort by' clause.

Note:

- The key of 'order by/sort by/distribute by' must be output column of Select statement. That is the column alias. In ODPS SQL analysis, 'order by/sort by/distribute by' operations are later that 'Select' operation. So they only accept the output column of Select statement as a key.

Subquery

The general 'Select' statement is to read data from a table, such as 'select column_1, column_2 ... from table_name'. But the object to be queried may be another 'Select' operation result, as follows:

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

Note:

- Alias must be used in subquery.

In ‘from’ clause, the subquery can be used as a table, which supports join operation with other tables or subquery. 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;
```

3.3.2.3 UNION ALL

Statement Format:

```
select_statement union all select_statement
```

Combine two or multiple data sets gotten from ‘Select’ operations to be a data set. If repeated rows exist in the result, all rows meeting the condition will be returned, not to remove duplicated data. Please note that ODPS SQL does not support combining two top query results and it must be rewritten to be a subquery format, as follows:

```
select * from sale_detail where region = 'hangzhou'

    union all

select * from sale_detail where region = 'shanghai';
```

Needs to be changed as:

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

Note:

- For ‘union all’ operation, the column number, column name and type of each subquery must be consistent. If the column names are not consistent, use column alias to solve this problem.
- Generally, ODPS allows 128 ways ‘union all’ at most. If exceeding this limit, report syntax error.

3.3.2.4 Join

JOIN Operations

ODPS JOIN supports multiple joints, but does not support Cartesian product, that is a link without ‘on’ condition.

Statement Format:

```
join_table:
table_reference join table_factor [join_condition]
```

```
| table_reference {left outer|right outer|full outer|inner} join table_reference join_condition

table_reference:
table_factor
| join_table

table_factor:
    tbl_name [alias]
    | table_subquery alias
  | ( table_references )

join_condition:
    on equality_expression ( and equality_expression )*
```

📖 **Note:**

- equality_expression is an equality expression.

‘Left join’ will return all records from left table (shop), even if there is no matching records in right table (sale_detail).

```
select a.shop_name as ashop, b.shop_name as bshop from shop a
    left outer join sale_detail b on a.shop_name=b.shop_name;
--Since the column shop_name exists both in table ‘shop’ and table ‘sale_detail’, the alias should be used in select clause to distinguish them.
```

‘Right outer join’ will return all records from right table, even if there is no matching records in left table, such as:

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

‘Full outer join’ will return all records from left table and right table. Such as:

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

If there is at least one matching record in the table, ‘inner join’ will return record. The keyword ‘inner’ can be ignored.

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

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

It’s only allowed to use ‘and’ to connect equal conditions and supports 16 ways of join operations at most. Only in MAPJOIN, unequal connections and using ‘or’ to connect multiple conditions are allowed.

```
select a.* from shop a full outer join sale_detail b on a.shop_name=b.shop_name
    full outer join sale_detail c on a.shop_name=c.shop_name;
--support multiple joints, and the maximum of ‘join’ examples is 16.
```

```
select a.* from shop a join sale_detail b on a.shop_name != b.shop_name;
--Report error because unequal connections connected by 'join' are not supported.
```

MAPJOIN HINT

When a big table needs to do 'join' operation with one or more small tables, 'mapjoin' can be used, which performance is much faster than ordinary 'join' operation. The basic principle of mapjoin is: SQL will load all specified small tables into the memory of a program which executes 'join' operation, so as to speed up the execution speed of 'join'. The following items should be noted to use 'mapjoin':

- The left table of 'left outer join' must be a big table.
- The right table of 'right outer join' must be a big tale.
- The left table or right table of 'inner join' can be considered as a big table.
- 'Full outer join' can not be used with 'mapjoin'.
- The 'mapjoin' statement supports a small table for the subquery.
- When using 'mapjoin' and needing to quote small tables or subquery, an alias needs to be quoted.
- In mapjoin, unequal connections and using 'or' to connect multiple conditions are allowed.
- At present, ODPS mapjoin supports up to six small tables, otherwise sytax error will be reported.
- If using 'mapjoin', then the memory size occupied by all small tables shall not exceed 512MB.

Next is a simple example:

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

ODPS SQL does not support using unequal expressions as 'on' condition in general 'join' clause and 'or' join condition. But in mapjoin, these operations are supported. For example:

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

3.3.3 Bulit-in Function

3.3.3.1 Arithmetic Function

ABS

Function Definition:

```
double abs(double number)
bigint abs(bigint number)
```

Use: return absolute value.

Parameter Description:

- Number: Double or bigint. The input is ‘bigint’ and return ‘bigint’; the inputs are ‘double’ and return ‘double’. If the input is ‘string’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown.

Return value: Double or bigint, which depends on the type of input parameter. If the input is null, return null.

📖 **Note:**

- When the value of input ‘bigint’ type is over the maximum value of ‘bigint’, return ‘double’ type. In this case, the precision may be lost.

Examples:

```
abs(null) =   null
abs(-1) = 1
abs(-1.2) = 1.2
abs("-2") = 2.0
abs(122320837456298376592387456923748) = 1.2232083745629837e32
```

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

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

ACOS

Function Definition:

```
double acos(double number)
```

Use: calculate inverse cosine function of specified number.

Parameter Description:

- Number: Double type, -1⩽number⩽1. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown.

Return value: Double type, the value is between 0 to π. If the number is NULL, return NULL.

Examples:

```
acos("0.87") = 0.5155940062460905
acos(0) = 1.5707963267948966
```

ASIN

Function Definition:

```
double asin(double number)
```

Use: arcsin function.

Parameter Description:

- number：Double type, -1≤number≤1. If the input is 'string' or 'bigint', it will be converted to 'double' by implicit conversion. If the input is other types, an error will be thrown.
- Return value: Double type, the value is between -π/2 to π/2. If the number is NULL, return NULL.

Example:

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

ATAN

Function Definition:

```
double atan(double number)
```

Use: arctan function.

Parameter Description:

- number：Double type. If the input is 'string' or 'bigint', it will be converted to 'double' by implicit conversion. If the input is other types, an error will be thrown.

Return value: Double type, the value is between -π/2 to π/2. If the number is NULL, return NULL.

Examples:

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

CEIL

Function Description:

```
bigint ceil(double value)
```

Use: return the minimum integer that is equal to or greater than the double value.

Parameter Description:

- Value: Double type. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown.

Return value: Bigint type. Any NULL input, return NULL.

Examples:

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

CONV

Function Definition:

```
string conv(string input, bigint from_base, bigint to_base)
```

Use: hexadecimal conversion function

Parameter Description:

- Input: an integer to be converted, represented by ‘string’. Accept the implicit conversion of bigint and double.
- from_base, to_base: decimal value, the acceptable values can be 2, 8, 10, 16. Accept the implicit conversion of bigint and double.

Return value: String type. Any NULL input, return NULL. It works by 64 bit precision in conversion process. Once overflowed, expection will be reported. If the input is a negative value (begin with ‘-’), expection will be reported. If the input is decimal, then convert it to an integer and do hexadecimal conversion, the decimal part will be discarded.

Example:

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

COS

Function Definition:

```
double cos(double number)
```

Use: cosine function; input is the radian value.

Parameter Description:

- Number: Double type. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown.

Return value: Double type. If the number is NULL, return NULL.

Example:

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

COSH

Function Definition:

```
double cosh(double number)
```

Use: hyperbolic cosine function.

Parameter Description:

- number： Double type. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown.

Return value: Double type. If the number is NULL, return NULL.

COT

Function Definition:

```
double cot(double number)
```

Use: cotangent function; input is the radian value.

Parameter Description:

- Number: Double type. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown. Return value: Double type. If the number is NULL, return NULL.

EXP

Function Definition:

```
double exp(double number)
```

Use: exponential function. Return the exponent value of number.

Parameter Description:

- Number: Double type. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown. Return value: Double type. If the number is NULL, return NULL.

FLOOR

Function Definition:

```
bigint floor(double number)
```

Use: return a maximum integer which is less than ‘number’.

Parameter Description:

- Number: Double type. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion.

Return value: return Bigint type. If the number is NULL, return NULL.

Example:

```
floor(1.2)=1
floor(1.9)=1
floor(0.1)=0
floor(-1.2)=-2
floor(-0.1)=-1
floor(0.0)=0
floor(-0.0)=0
```

LN

Function Definition:

```
double ln(double number)
```

Use: return the natural logarithm of the number.

Parameter Description:

- Number: Double type. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the number value is NULL, return NULL. If the number is negative or zero, expection will be thrown. Return value: Double ttype.

LOG

Function Definition:

```
double log(double base, double x)
```

Use: return the logarithm of x whose base number is ‘base’.

Parameter Description:

- ‘base’: Double type. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown.
- ‘x’: Double type. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown.

Return value: the logarithm value of Double type. If ‘base’ or ‘x’ is NULL, return NULL. If one of ‘base’ and ‘x’ is negative or zero, it will cause abnormality. If ‘base’ is 1, it will also cause abnormality.

POW

Function Definition:

```
double pow(double x, double y)
```

Use: return x to the yth power, that is x^y.

Parameter Description:

- 'x': Double type. If the input is 'string' or 'bigint', it will be converted to 'double' by implicit conversion. If the input is other types, an error will be thrown.
- 'y': Double type. If the input is 'string' or 'bigint', it will be converted to 'double' by implicit conversion. If the input is other types, an error will be thrown.

Return value: Double type. If x or y is NULL, return NULL.

RAND

Function Definition:

```
double rand(bigint seed)
```

Use: return a random number (that changes from row to row). Specifiying the seed will make sure the generated random number sequence is deterministic. Return value range is from 0 to 1.

Parameter Description:

- Seed: Bigint type, random number seed, to determine starting values of the random number sequence. Return value: Double type.

ROUND

Function Definition:

```
double round(double number, [bigint decimal_places])
```

Use: Four to five homes to the specified decimal point position.

Parameter Description:

- Number: Double type. If the input is 'string' or 'bigint', it will be converted to 'double' by implicit conversion. If the input is other types, an error will be thrown.
- decimal_place: a Bigint type constant, four to five homes to the decimal point position. If it is other type, exception will be thrown. If you omit it, it indicates four to five homes into the single digit. The default value is 0. Return value: return Double type. If 'number' or 'decimal_places' is NULL, return NULL.

Note:

- Decimal_places can be negative. The negative will be counted from decimal point to left and do not keep the decimal part. If decimal_places is greater than the length of the integer part, return 0.

Examples:

```
round(125.315) = 125.0
round(125.315, 0) = 125.0
round(125.315, 1) = 125.3
round(125.315, 2) = 125.32
round(125.315, 3) = 125.315
round(-125.315, 2) = -125.32
round(123.345, -2) = 100.0
round(null) = null
round(123.345, 4) = 123.345
round(123.345, -4) = 0.0
```

SIN

Function Definition:

```
double sin(double number)
```

Use: sine function, the input is the radian value.

Parameter Description:

- Number: Double type. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown.Return value: Double type. If the number is NULL, return NULL.

SINH

Function Definition:

```
double sinh(double number)
```

Use: hyperbolic sine function

Parameter Description:

- Number: Double type. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown.Return value: Double type. If the number is NULL, return NULL.

SQRT

Function Definition:

```
double sqrt(double number)
```

Use: calculate square root.

Parameter Description:

- Number: Double type, must be greater than 0. If it is less than 0, expection will be caused. If the input is ‘string’ or ‘bigint’, it will be converted to ‘double’ by implicit conversion. If the input is other types, an error will be thrown.Return value: Double type. If the number is NULL, return NULL.

TAN

Function Definition:

```
double tan(double number)
```

Use: Tangent function, the input is the radian value.

Parameter Description:

- Number: Double type. If the input is 'string' or 'bigint', it will be converted to 'double' by implicit conversion. If the input is other types, an error will be thrown.Return value: Double type. If the number is NULL, return NULL.

TANH

Function Definition:

```
double tanh(double number)
```

Use: hyperbolic tangent function.

Parameter Description:

- Number: Double type. If the input is 'string' or 'bigint', it will be converted to 'double' by implicit conversion. If the input is other types, an error will be thrown.Return value: Double type. If the number is NULL, return NULL.

TRUNC

Function Definition:

```
double trunc(double number[, bigint decimal_places])
```

Use: intercept the input number to a specified decimal point place.

Parameter Description:

- Number: Double type. If the input is 'string' or 'bigint', it will be converted to 'double' by implicit conversion. If the input is other types, an error will be thrown.
- decimal_places: a Bigint type constant, the decimal point place to intercept the number. Other types will be converted to 'bigint'. If this parameter is omitted, default to intercept to single digit. Return value: Double type. If number or decimal_places is NULL, return NULL.

Note:

- The part to be intercepted off will be supplemented with 0.
- decimal_places: can be negative. The negative will be intercepted from decimal point to left and do not keep the decimal part. If decimal_places is greater than the length of the integer part, return 0.

Examples:

```
trunc(125.815) = 125.0
trunc(125.815, 0) =125.0
trunc(125.815, 1) = 125.8
trunc(125.815, 2) = 125.81
trunc(125.815, 3) = 125.815
trunc(-125.815, 2) = -125.81
trunc(125.815, -1) = 120.0
trunc(125.815, -2) = 100.0
trunc(125.815, -3) = 0.0
trunc(123.345, 4) = 123.345
trunc(123.345, -4) = 0.0
```

3.3.3.2 Functions to Process String

CHAR_MATCHCOUNT

Function Definition:

```
bigint char_matchcount(string str1, string str2)
```

Use: it is used to calculate total times for which each character in str1 appears in str2.

Parameter Description:

- str1, str2: String type, must be effective UTF-8 strings. If there is invalid character in matching process, then return a negative value.

Return value: Bigint type. Any NULL input, return NULL.

Example:

```
char_matchcount('abd', 'aabc') = 2
--Two strings 'a', 'b' in str1 appear in str2.
```

CHR

Function Definition:

```
string chr(bigint ascii)
```

Use: convert the specified ASCII code ‘ascii’ into character.

Parameter Description:

- ‘ascii’: Bigint type ASCII value. If the input is ‘string’ or ‘double’, it will be converted to ‘bigint’ by implicit conversion. If the input is other types, an error will be thrown.

Return value: String type. The parameter value range is 0~255. The expection will be thrown if exceeding this range. If the input is NULL, return NULL.

CONCAT

Function Definition:

```
string concat(string a, string b...)
```

Use: the return value is a result after connecting all strings.

Parameter Description:

- ‘a’, ‘b’…: string type. If the input is ‘bigint’ or ‘double’ or ‘datetime’, it will be converted to ‘string’ by implicit conversion. If the input is other types, an error will be thrown.

Return value: String type. If there is no parameter or a certain parameter is NULL, return NULL.

Examples:

```
concat('ab', 'c') = 'abc'
concat() = NULL
concat('a', null, 'b') = NULL
```

INSTR

Function Definition:

```
bigint instr(string str1, string str2[, bigint start_position[, bigint nth_appearance]])
```

Use: calculate the position that a substring str2 is located in str1.

Parameter Description:

- str1: String type, a string to be searched. If the input is ‘bigint’ or ‘double’ or ‘datetime’, it will be converted to ‘string’ by implicit conversion. If the input is other types, an error will be thrown.
- Str2: String type, a substring to be searched. If the input is ‘bigint’ or ‘double’ or ‘datetime’, it will be converted to ‘string’ by implicit conversion. If the input is other types, an error will be thrown.
- start_position: Bigint type, for other types, expection will be thrown. It indicates which character of str1 will be searched from and the default staring position is the first character position 1. If it is less than 0, it will cause abnormality.
- nth_appearance: Bigint type, greater than 0. It indicates the nth_appearance position of the substring in string. If nth_appearance is other type or 0, expection will be thrown.

Return value: Bigint type.

Note:

- If str2 is not found in str1, return 0.
- Any input parameter is NULL, return NULL
- If str2 is NULL and always can be matched successfully, so instr (‘abc’, ‘’) will return 1.

Examples:

```
instr('Tech on the net', 'e') = 2
instr('Tech on the net', 'e', 1, 1) = 2
instr('Tech on the net', 'e', 1, 2) = 11
instr('Tech on the net', 'e', 1, 3) = 14
```

IS_ENCODING

Function Definition:

```
boolean is_encoding(string str, string from_encoding, string to_encoding)
```

Use: judge whether the input string ‘str’ can be changed into a character set ‘to_encoding’ from a specified character set ‘from_encoding’. It can be used to judge whether the input is garbled. The general use is to set ‘from_encoding’ to be 'UTF-8' and ‘to_encoding’ to be 'GBK'.

Parameter Description:

- Str: String type, if the input is NULL, return NULL. The empty string can be considered belong to any character set.
- from_encoding, to_encoding: String type, source and destination character sets. If the input is NULL, return NULL.

Return value: Boolean type. If ‘str’ can be converted successfully, return true, otherwise, return false.

Examples:

```
is_encoding('测试', 'utf-8', 'gbk') = true
is_encoding('測試', 'utf-8', 'gbk') = true
--There are these two traditional Chinese characters in GBK stock.
is_encoding('測試', 'utf-8', 'gb2312') = false
-- The word stock ‘gb2312’ does not contain these two Chinese characters.
```

LENGTH

Function Definition:

```
bigint length(string str)
```

Use: return the length of a string.

Parameter Description:

- ‘str’: String type. If the input is ‘bigint’ or ‘double’ or ‘datetime’, it will be converted to ‘string’ by implicit conversion. If the input is other types, an error will be thrown.

Return value: Bigint type. If ‘str’ is NULL, return NULL. If ‘str’ is non UTF-8 coding format, return -1.

Example:

```
length('hi! 中国') = 6
```

LENGTHB

Function Definition:

```
bigint lengthb(string str)
```

Use: return the length of ‘str’, which unit is byte.

Parameter Description:

- 'str': String type. If the input is 'bigint' or 'double' or 'datetime', it will be converted to 'string' by implicit conversion. If the input is other types, an error will be thrown.

Return value: Bigint type. If 'str' is NULL, return NULL.

Example:

```
lengthb('hi! 中国') = 10
```

MD5

Function Definition:

```
string md5(string value)
```

Use: calculate the md5 value of input string.

Parameter Description:

- Value: String type. If the input is 'bigint' or 'double' or 'datetime', it will be converted to 'string' by implicit conversion. If the input is other types, an error will be thrown. If the input is NULL, return NULL.

Return value: String type.

REGEXP_EXTRACT

Function Definition:

```
string regexp_extract(string source, string pattern[, bigint occurrence])
```

Use: Split the string source according to pattern regular expression rules, and return the charactors of the occurrence group.

Parameter Description:

- Source: String type, a string to be searched.
- Pattern: a String type constant. If pattern is a null string, expection will be thrown. If 'group' is not specified in pattern, expection will also be thrown.
- Occurrence: a Bigint type constant, must be greater than 0 or equal to 0. If it is other type or less than 0, expection will be thrown. If not specified, the default value is 1, which indicates returning the first group. If occurrence = 0, return a substring which meets a whole pattern requirement.

Return value: String type. Any input is NULL, return NULL.

Examples:

```
regexp_extract('foothebar', 'foo(.*?)(bar)', 1) = the
regexp_extract('foothebar', 'foo(.*?)(bar)', 2) = bar
regexp_extract('foothebar', 'foo(.*?)(bar)', 0) = foothebar
regext_extract('8d99d8', '8d(\\d+)d8') = 99
-- If regular SQL is submitted on ODPS, two "\" must be used as the shift character.
```

```
regexp_extract('foothebar', 'foothebar')
--The expection is thrown. ‘group’ is not specified in ‘pattern’.
```

REGEXP_INSTR

Function Definition:

```
bigint regexp_instr(string source, string pattern[,
    bigint start_position[, bigint nth_occurrence[, bigint return_option]])
```

Use: return the start position/end position of the substring, which is matched with source from start_position and nth_occurrence with pattern. Any input parameter is null, return null.

Parameter Description:

- Source: String type, to be searched.
- Pattern: a String type constant. If ‘pattern’ is null, expection will be thrown.
- start_position: Bigint type constant, the start position of search. If it is not specified, default value is 1. If it is other type or a value which is less than or eauql to 0, expection will also be thrown.
- nth_occurrence: a Bigint type constant. If not specified, the default value is 1, which indicates the first appearance position in searching course. It it is less than or equal to 0 or other type, expection will be thrown.
- return_option: a Bigint type constant. Its value is 0 or 1. If it is other type or a unallowed value, expection will be thrown. 0 indicates returning the start position of the matching. 1 indicates returning the end position of the matching.

Return value: Bigint type, the start or end position of matched substring in source specified by return_option.

Examples:

```
regexp_instr("i love www.taobao.com", "o[[:alpha:]]{1}", 3, 2) = 14
```

REGEXP_REPLACE

Function Definition:

```
string regexp_replace(string source, string pattern, string replace_string[, bigint occurrence])
```

Use: replace the substring in source which is matched ‘pattern’ for nth occurrence to be a specified string ‘replace_string’ and then return.

Parameter Description:

- Source: String type, a string to be replaced.
- Pattern: String type constant. The pattern to be matched. If it is null, expection will be thrown.
- replace_string: String type, the string after replacing matched pattern.
- Occurrence: Bigint type constant, must be greater than or equal to 0, which

indicates replacing nth matching to be ‘replace_string’. If it is 0, it indicates all matched substrings have been replaced. If it is other type or less than 0, expection will be thrown. It can be default, and the default is 0.

Retrurn value: String type. When referencing a group which is not existent, do not replace the string. If the input source, pattern or occurrence parameter is NULL, return NULL. If replace_string is NULL and pattern is matched, return NULL. If replace_string is NULL but is not matched with pattern, return the source string.

Example:

```
regexp_replace("123.456.7890", "([[:digit:]]{3})\\.([[:digit:]]{3})\\.([[:digit:]]{4})",
    "(\\1)\\2-\\3", 0) = "(123)456-7890"
regexp_replace("abcd", "(.)", "\\1 ", 0) = "a b c d "
regexp_replace("abcd", "(.)", "\\1 ", 1) = "a bcd"
regexp_replace("abcd", "(.)", "\\2", 1) = "abcd"
-- Only a group is defined in pattern and the referenced second group is not existent.
-- Please avoid this. The result to reference nonexistent group is not defined.
regexp_replace ("abcd", "(.*)(.)$", "\\2", 0) = "d"
regexp_replace("abcd", "a", "\\1", 0) = "bcd"
-- There is no group definition in pattern, so '\1' references a nonexistent group.
--Please avoid this. The result to reference nonexistent group is not defined.
```

REGEXP_SUBSTR

Function Definition:

```
string regexp_substr(string source, string pattern[, bigint start_position[, bigint nth_occurrence]])
```

Use: starting from start_position, find the substring in ‘source’ which matches specified pattern for the nth occurrence.

Parameter Description:

- Source: String type, string to be searched.
- Pattern: a String type constant. The pattern to be matched. If it is null, expection will be thrown.

start_position: a Bigint type constant, must be greater than 0. If it is other type or a value which is less than or eauql to 0, expection will be thrown. If it is not specified, default value is 1, which indicates matching from the first character of ‘source’.

nth_occurrence: a Bigint type constant, must be greater than 0. If not specified, the default value is 1, which indicates returning the substring of first matching.

Return value: String type. Any input partameter is NULL, return NULL. If thers is no matching record, return NULL.

Example:

```
regexp_substr ("I love aliyun very much", "a[[:alpha:]]{5}") = "aliyun"
regexp_substr('I have 2 apples and 100 bucks!', '[[:blank:]][[:alnum:]]*', 1, 1) = " have"
regexp_substr('I have 2 apples and 100 bucks!', '[[:blank:]][[:alnum:]]*', 1, 2) = " 2"
```

REGEXP_COUNT

Function Definition:

```
bigint regexp_count(string source, string pattern[, bigint start_position])
```

Use: count the occurrence that the substring matches specified pattern starting from start_position in ‘source’.

Parameter Description:

Source: String type, the string to be searched. If it is other type, report the error.

Pattern: String type constant. The pattern to be matched. If it is null string or other data type, expection will be thrown.

start_position: Bigint type constant, must be greater than 0. If it is other data type or a value which is less than or eauql to 0, expection will be thrown. If it is not specified, default value is 1, which indicates matching from the first character of source.

Return value: Bigint type. If there is no matching, return 0. If any input parameter is null, return null.

Example:

```
regexp_count('abababc', 'a.c') = 1
regexp_count('abcde', '[[:alpha:]]{2}', 3) = 1
```

SPLIT_PART

Function Definition:

```
string split_part(string str, string separator, bigint start[, bigint end])
```

Use: split the string ‘str’ according to the separator and return the substring from nth start part to nth end part.

Parameter Description:

‘str’: String type, the string to be split. If it is ‘bigint’ or ‘double’ or ‘datetime’, it will be converted to ‘string’ in implicit conversion. If it is other data type, expection will be reported.

‘separator’: a String type constant, the separator used to split the string. It can be a character or a string. If it is other data type, expection will be caused.

‘start’: a Bigint type constant, must be greater than 0. If it is not a constant or other data type, expection will be thrown. It indicated the start numer of return part (start from 1). If the end is not specified, return the part specified by ‘start’.

‘end’: a Bigint type constant, must be greater than or equal to ‘start’, otherwise expection will be thrown. It refers to the end number of return part. If it is not a constant or is other data type, expection will also be thrown. It can be omitted, which indicates the last part.

Return value: String type. If any parameter is null, return null. If ‘separator’ is empty string, return the source string ‘str’.

📖 **Note:**

- If ‘separator’ does not exist in ‘str’ and ‘start’ is specified to be 1, return whole string. If the input is an empty string, the output is also an empty string.
- If the value of ‘start’ is greater than actual part quantity after split, return empty string. For example, the string is split to 6 parts but ‘start’ is greater than 6, return empty string.
- If ‘end’ is greater than ‘part’ quantity, take it as the part quantity.

Example:

```
split_part('a,b,c,d', ',', 1) = 'a'
split_part('a,b,c,d', ',', 1, 2) = 'a,b'
split_part('a,b,c,d', ',', 10) =   ''
```

SUBSTR

Function Definition:

```
string substr(string str, bigint start_position[, bigint length])
```

Use: return the substring of ‘str’ from start_position with the given length. E.g. substr ('foobar', 4, 2) results in 'ba'.

Parameter Description:

‘str’: String type. If the input is ‘bigint’ or ‘double’ or ‘datetime’, it will be converted to ‘string’ in implicit conversion. If it is other data type, report the expection.

‘start_position’: Bigint type. When the value of ‘start_position’ is negative, it indicates that the start position is from the string end toward forward. The last character is -1 and the start position is 1. If it is other data type, expection will be thrown.

‘length’: Bigint type, must be greater than 0. If it is other type or less than 0, expection will be thrown.

Return value: String type. If any input is NULL, return NULL.

📖 **Note:**

- If the length is omitted, return the substring from start to end.

Example:

```
substr("abc", 2) = "bc"
substr("abc", 2, 1) = "b"
```

TOLOWER

Function Definition:

```
string tolower(string source)
```

Use: input the lowercase string corresponding to the English string ‘source’.

Parameter Description:

Source: String type. If the input is ‘bigint’ or ‘double’ or ‘datetime’, it will be converted to ‘string’ in implicit conversion. If it is other data type, expection will be thrown.

Return value: String type. If the input is NULL, return NULL.

Example:

```
tolower("aBcd") = "abcd"
tolower("哈哈 Cd") = "哈哈 cd"
```

TOUPPER

Function Definition:

```
string toupper(string source)
```

Use: output the uppercase string corresponding to the English string ‘source’.

Parameter Description:

Source: String type. If the input is ‘bigint’, ‘’double’ or ‘datetime’, it will be converted to ‘string’ in implicit conversion. If it is other data type, expection will be reported.

Return Value: String type. If the input is NULL, return NULL.

Example:

```
toupper("aBcd") = "ABCD"
toupper("哈哈 Cd") = "哈哈 CD"
```

TO_CHAR

Function Definition:

```
string to_char(boolean value)
string to_char(bigint value)
string to_char(double value)
```

Use: convert ‘Boolean’ type, ‘bigint’ type or ‘double’ type to corresponding ‘string’ type.

Parameter Description:

Value: ‘boolean’ type, ‘bigint’ type or ‘double’ type is acceptable. If it is other data type, expection will be thrown. For the formatting of ‘datetime’ type, please refer to another same name function in TO_CHAR.

Return value: String type. If the input is NULL, return NULL.

Example:

```
to_char(123) = '123'
to_char(true) = 'TRUE'
to_char(1.23) = '1.23'
to_char(null) = NULL
```

TRIM

Function Definition:

```
string trim(string str)
```

Use: remove left space and right space for the input string ‘str’.

Parameter Description:

‘str’: String type. If the input is ‘bigint’, ‘’double’ or ‘datetime’, it will be converted to ‘string’ in implicit conversion. If it is other data type, expection will be reported.

Return value: String type. If the input is NULL, return NULL.

3.3.3.3 Functions to Process Date

DATEADD

Function Definition:

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

Use: modify the value of date according to specified unit ‘datepart’ and specified scope ‘delta’.

Parameter Description:

‘date’: Datetime type, value of date. If the input is string type, it will be converted to ‘datetime’ type by implicit conversion. If it is other type, expection will be thrown.

‘delta’: Bigint type, date scope to be modified. If the input is ‘string’ type or ‘double’ type, it will be converted to ‘bigint’ type by implicit conversion. If it is other data type, expection will be caused. If ‘delta’ is greater than 0, do ‘add’ operation, otherwise do ‘minus’ operation.

‘datepart’: a String type constant. This field value follows ‘string’ and ‘datetime’ type conversion agreement, that is, ‘yyyy’ indicates year; ‘mm’ indicates month… 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, expection will be thrown.

Return value: Datetime type. If any input is NULL, return NULL.

Note:

- While increasing or decreasing ‘delta’ according to specified unit, it will cause the carry or back space for higher unit. Day, month, hour, minute and 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’ will 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 there is no special description, 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.

Example:

```
If trans_date = 2005-02-28 00:00:00：
dateadd(trans_date, 1, 'dd') = 2005-03-01 00:00:00
--Add one day. The result is beyond the last day in February. The actual value is the first day of next month.
dateadd(trans_date, -1, 'dd') = 2005-02-27 00:00:00
--Minus one day
dateadd(trans_date, 20, 'mm') = 2006-10-28 00:00:00
--Add 20 months. The month spillover is caused and the year is added ‘1’.

If trans_date = 2005-02-28 00:00:00, dateadd(transdate, 1, 'mm') = 2005-03-28 00:00:00
If trans_date = 2005-01-29 00:00:00, dateadd(transdate, 1, 'mm') = 2005-02-28 00:00:00
--There is no 29th 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
```

In ODPS SQL, the datetime type has no direct constant representation, the following use way is wrong:

```
select dateadd(2005-03-30 00:00:00, -1, 'mm') from tbl1;
```

If you must describe the datetime type constant, please 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)
```

Use: calculate the difference between two datetime date1 and date2 in specified time unit ‘datepart’.

Parameter Description:

datet1, date2: Datetime type, minuend and meiosis. If the input is ‘string’, it will be converted to ‘datetime’ by implicit conversion. If it is other data type, expection will be thrown.

‘datepart’: a String type constant. The extensional date format is supported. If ‘datepart’ does not meet specified format or is other data type, it will cause expection.

Return value: Bigint type. Any input parameter is NULL, return NULL.

Note:

- The lower unit part will be cut off according to ‘datepart’ in calculation process and then calculate the result.

Examples:

```
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

Function Definition:

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

Use: extract the value of specified time unit ’datepart’ in ‘date’.

Parameter Description:

‘date’: Datetime type. If the input is ‘string’ type, it will be converted to ‘datetime’ type. If it is other data type, expection will be thrown.

‘datepart’: String type constant. The extensional date format is supported. If ‘datepart’ does not meet specified format or is other data type, it will cause expection.

Return value: Bigint type. If any input is NULL, return NULL.

Example:

```
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)
```

Use: 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 will be converted to ‘datetime’ type. If it is other data type, expection will be thrown.

‘datepart’: String type constant. The extensional date format is supported. If ‘datepart’ does not meet specified format or is other data type, it will cause expection.

Return value: Datetime type. If any input is NULL, return NULL.

Examples:

```
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)
```

Use: 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 will be converted to ‘bigint’ type by implicit conversion.

Return value: Datetime type date value. If ‘unixtime’ is NULL, return NULL.

Examples:

```
from_unixtime(123456789) = 2009-01-20 21:06:29
```

GETDATE

Function Definition:

```
datetime getdate()
```

Use: get present system time. Use GMT+8 as ODPS standard time.

Return value: Datetime type, return present date and time.

Note:

- In an ODPS SQL task (executed in a distributed manner), ‘getdate’ always returns a fixed value. The return result will be any time in ODPS SQL execution period and the precision of time accurates to seconds.

ISDATE

Function Definition:

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

Use: 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 will be converted to ‘string’ type. If it is other data type, expection will be reported.

Format: a String type constant. The extensional date format is not supported. If it is other data type or the format which is not supported, expection will be thrown. If there are redundant format strings appearing in ‘format’, then get the datatime value corresponding to the first format string, other strings will be taken as seperators. For example, isdate (‘1234-yyyy’, ‘yyyy-yyyy’) will return ‘TRUE’.

Return value: Boolean type. If any parameter is NULL, return NULL.

LASTDAY

Function Definition:

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

Use: 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 will be converted to ‘datetime’ type. If it is other data type, expection will be reported.

Return value: Datetime type. If the input is NULL, return NULL.

TO_DATE

Function Definition:

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

Use: conver a string ‘date’ to the datetime value according to specified format.

Parameter Description:

Date: String type, date value to be converted. If the input is ‘bigint’, or ‘double’ or ‘datetime’, it will be converted to ‘string’ type by implicit conversion. If it is other data type or null, expection will be thrown.

Format: String type constant, date format. If it is not a constant or is other data type, the expection will be caused. The field ‘format’ does not support extensional format and other characters will be ignored as invaid characters in analysis process. The parameter ‘format’ should contain ‘yyyy’ at least; otherwise the expecion will be caused. If there are redundant format strings appearing in ‘format’, then get the datatime value corresponding to the first format string, other strings will be taken as seperators. For example, to_date (‘1234-2234’, ‘yyyy-yyyy’) will return ‘1234-01-01 00:00:00’.

Return value: Datetime type. If any input is NULL, return NULL.

Examples:

```
to_date('ALIBABA 2010-12*03', 'ALIBABA yyyy-mm*dd') = 2010-12-03 00:00:00
to_date('20080718', 'yyyymmdd') = 2008-07-18 00:00:00


to_date('2008718', 'yyyymmdd')
```

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

TO_CHAR

Function Definition:

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

Use: convert the ‘date’ of datetime type to a string according to specified format.

Parameter Description:

Date: Datetime type, the date value to be converted. If the input is ‘string’ type, it will be converted to ‘datetime’ type by implicit conversion. If it is other data type, expection will be thrown.

Format: String type constant. If it is not a constant or is other data type, the expection will be caused. In ‘format’, the date format part will be replaced with the corresponding data and other characters are output directly.

Return value: String type. Any input parameter is NULL, return NULL.

```
to_char('ALIBABA 2010-12*03', 'ALIBABA yyyy-mm*dd') = '2010-12-03 00:00:00'
to_char('20080718', 'yyyymmdd') = '2008-07-18 00:00:00'

to_char('ALIBABA 2010-12*3', 'ALIBABA yyyy-mm*dd')
-- Format is not compatible and expection is thrown.
to_char('2010-24-01', 'yyyy')
-- Format is not compatible and expection is thrown.
to_char('2008718', 'yyyymmdd')
-- Format is not compatible and expection is thrown.
```

UNIX_TIMESTAMP

Function Definition:

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

Use: 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 will be converted to ‘datetime’ type and involved in calculation. If it is other type, expection will be thrown.

Return value: Bigint type, it indicates UNIX format date value. If ‘date’ is NULL, return NULL.

WEEKDAY

Function Definition:

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

Use: return the nth day of present week corresponding to the date.

Parameter Description:

Date: Datetime type. If the input is 'string' type, it will be converted to 'datetime' type and then involved in operation. If it is other date type, expection will be thrown.

Return value: Bigint type. If the input parameter is NULL, return NULL. Monday is considered as the first day of a week and corresponding return value is 0. Other dates are in increasing order. If the date is Sunday, return 6.

WEEKOFYEAR

Function Definition:

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

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

Parameter Description:

Date: Datetime type. If the input is 'string' type, it will be converted to 'datetime' type and then involved in operation. If it is other date type, expection will be thrown.

Return value: Bigint type. If the input is NULL, return NULL.

3.3.3.4 Window Function

In ODPS 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 can not be used with the same level aggregation function together. At present, in an ODPS SQL statement, you can use at most 5 window functions.

Window Function Synax

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

The part 'partition by' is used to specify open window columns. The rows which partition columns have the same values are considered in the same window. At this stage, the same window can contain at most 100000000 rows of data; otherwise an error will be reported in running time.

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. There are two ways: **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 and sum.

COUNT

Function Definition:

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

Use: 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 columes 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 valiue: Bigint type.

🕮 **Note:**

- If the keyword 'distinct' has been specified, the 'order by' clause can not 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          | -- there are two records 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])
```

Use: 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 will be converted to 'double' type by implicit conversion and involed in operation. If it is other data type, expection will be thrown. If this value is NULL, this row is not involed in calculation. If the data type is Boolean, it is not allowed involing in calculation.

partition by col1[, col2]…: speficied 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 return the cumulative average from start row to current row in current window.

Return value: Double type.

Note:

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

MAX

Function Definition:

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

Use: calculate the maximum value.

Parameter Description:

Expr: Any types expect ‘Boolean’. If the value is NULL, this row is not involved in calculation. If the keyword ‘distinct’ is specified, it indicates taking the max value of unique value.

partition by col1[, col2…]: speficied 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 can not be used.

MIN

Function Definition:

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

Use: calculate the minimum value of the column.

Parameter Description:

Expr: Any types expect ‘Boolean’. If the value is NULL, this row will not be involved in calculation. If the keyword ‘distinct’ is specified, it indicates taking the minimum value of a unique value.

partition by col1[, col2..]: speficied 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 can not be used.

MEDIAN

Function Definition:

```
double median(double number) over(partition by col1[, col2…])
```

Use: calculate the median.

Parameter Description:

Number: Double type. If the input is 'string' or 'bigint' type, it will be converted to 'double' type and involved in operation. If it is other data type, expection will be thrown. If the input is null, ignore it.

partition by col1[, col2…]: Speficied 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])
```

Use: calculation population standard deviation.

Parameter Description:

Expr: Double type. If the input is 'string' or 'bigint' type, it will be converted to 'double' type and involved in operation. If it is other data type, expection will be thrown. 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..]: speficied 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: Double type.

📖 **Note:**

- If the keyword 'distinct' has been specified, the 'order by' clause can not be used.

STDDEV_SAMP

Function Definition:

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

Use: calculate sample standard deviation.

Parameter Description:

Expr: Double type. If the input is ‘string’ or ‘bigint’ type, it will be converted to ‘double’ type and involved in operation. If it is other data type, expection will be thrown. 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..]: speficied 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: Double type.

🕮 **Note:**

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

SUM

Function Definition:

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

Use: calculate the sum of elements.

Parameter Description:

Expr: Double type. If the input is ‘string’ or ‘bigint’ type, it will be converted to ‘double’ type and involved in operation. If it is other data type, expection will be thrown. 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..]: speficied 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 ‘double’ type or ‘string’ type, return ‘double’ type.

🕮 **Note:**

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

DENSE_RANK

Function Definition:

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

Use: calculate dense rank.

Parameter Description

partition by col1[, col2..]: speficied 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.

RANK

Function Definition:

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

Use: calculate the rank. The ranking of the same row data with col2 will drop.

Parameter Description:

partition by col2[, col2..]: speficy 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.

LAG

Function Definition:

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

Use: 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..]: speficy columns to use window function.

order by col1 [asc|desc], col2[asc|desc]: specify the order method for return result.

Return value: the same with ‘expr’.

LEAD

Function Definition:

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

Use: 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’ 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.

partition by col1[, col2..]: speficy columns to use window function.

order by col1 [asc|desc], col2[asc|desc]: specify the order method for return result.

Return value: the same with ‘expr’.

PERCENT_RANK

Function Definition:

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

Use: calculate relative ranking of a certain row in a group of data.

Parameter Description:

partition by col1[, col2..]: speficy columns to use window function.

order by col1 [asc|desc], col2[asc|desc]: specify the value which the ranking is based on.

Return value: 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 number in a single window can not exceed 10,000,000.

ROW_NUMBER

Function Definition:

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

Use: calculate the row number, beginning from 1.

Parameter Description:

partition by col1[, col2..]: speficy columns to use window function.

order by col1 [asc|desc], col2[asc|desc]: specify the order method for return result.

Return value: Bigint type.

CLUSTER_SAMPLE

Function Definition:

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

Use: 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 there are x rows 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 there is y parts records, return value is true.). If 'y' is NULL, return NULL.

'partition by col1[, col2]': speficy columns to use window function.

Return value: Boolean type.

Example:

If there are two columns 'key' and 'value' in the table 'test_tbl'; 'key' is grouping field. The corresponding values of key have 'groupa' and 'groupb' and 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   |
+------------+--------------------+
```

In orde to select 10% values from each group, the following ODPS 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 | -1.34764165478145   |
| groupb | -0.725703978381499 |
| groupb | 2.4704244205196       |
+-----+-----------------------+
```

3.3.3.5 Aggregation Function

The relationship between input and output of aggregation function is many-to-one relationship. That is to aggregate multiple input records into an output record. It can be used with ‘group by’ clause in SQL.

COUNT

Function Definition:

```
bigint count([distict|all] value)
```

Use: count the record number.

Parameter Description:

distinct|all: specify whether to remove duplicate records in the count process. The default is ‘all’, to count all records. If the field ‘distinct’ is specified, it indicates taking the unique count value.

Value: any type. If the value is NULL, corresponding row will not be involved in count. Count (), return all rows.

Return value: Bigint type.

Example:

```
--if the table 'tbla' has the column 'col1' and the data type is bigint.
+------+
| COL1 |
+------+
| 1    |
+------+
| 2    |
+------+
| NULL |
+------+

select count(*) from tbla;    --value is 3.
select count(col1) from tbla;    --value is 2
```

Aggregation function can be used with 'group by' clause. For example, suppose that the table 'test_src' is existent and has two columns: key string, value double.

```
--The data of test_src is shown as follows:
+-----+-------+
| key | value |
+-----+-------+
| a   | 2.0   |
+-----+-------+
| a   | 4.0   |
+-----+-------+
| b   | 1.0   |
+-----+-------+
| b   | 3.0   |
+-----+-------+
--Now execute following sentence and get the result::

select key, count(value) as count from test_src group by key;

+-----+-------+
| key | count |
+-----+-------+
| a   | 2     |
+-----+-------+
| b   | 2     |
+-----+-------+
-- The aggregation function is to do aggregation calation for the values which has the same key value.
```

AVG

Function Definition:

```
double avg(double value)
```

Use: calculate average.

Parameter Description:

Value: Double type. If the input is ‘string’ type or ‘bigint’ type, it will be converted to ‘double’ type by implicit conversion and involed in operation. If it is other data type, expection will be thrown. If this value is NULL, corresponding row is not involed in calculation. Boolean type is not allowed involving in calculation.

Return value: Double type.

Example:

```
--If the table ‘tbla’ has a column ‘value’ and its data type is Bigint.

+-------+
| value |
+-------+
| 1     |
| 2     |
| NULL  |
+-------+

-- the avg of this column is: (1+2)/2=1.5

select avg(value) as avg from tbla;

+------+
| avg  |
+------+
| 1.5  |
+------+
```

MAX

Function Definition:

```
max(value)
```

Use: calculate the maximum value.

Parameter Description:

Value: can be any data type. If the column value is NULL, corresponding row will not be involved in operation. Boolean type is not allowed involving in operation.

Return value: the same type with ‘value’.

Example:

```
--If the table ‘tbla’ has a column ‘clo1’ and its data type is Bigint.
```

```
+------+
| col1 |
+------+
| 1      |
+------+
| 2      |
+------+
| NULL |
+------+
select max(value) from tbla; --return value is 2.
```

MIN

Function Definition:

```
MIN(value)
```

Use: calculate the minimum value.

Parameter Description:

Value: can be any data type. If the column value is NULL, this corresponding row will not be involved in operation. Boolean type is not allowed involving in operation.

Example:

```
--If the table 'tbla' has a column 'value' and its data type is Bigint.

+------+
| value|
+------+
| 1      |
+------+
| 2      |
+------+
| NULL |
+------+
select min(value) from tbla; --return value is 1.
```

MEDIAN

Function Definition:

```
double median(double number)
```

Use: calculate the median.

Parameter Description:

Number: Double type. If the input is 'string' or 'bigint' type, it will be converted to 'double' type and involved in operation. If it is other data type, expection will be thrown. If the input value is NULL, this row will be ignored.

Return value: Double type.

STDDEV

Function Definition:

```
double stddev(double number)
```

Use: calculate population standard deviation.

Parameter Description:

Number: Double type. If the input is ‘string’ or ‘bigint’ type, it will be converted to ‘double’ type and involved in operation. If it is other data type, expection will be thrown. If the input value is NULL, ignore this row.

Return value: Double type.

STDDEV_SAMP

Function Definition:

```
double stddev_samp(double number)
```

Use: calculate sample standard deviation.

Parameter Description:

Number: Double type. If the input is ‘string’ or ‘bigint’ type, it will be converted to ‘double’ type and involved in operation. If it is other data type, expection will be thrown. If the input value is NULL, ignor this row.

Return value: Double type.

SUM

Function Definition:

```
sum(value)
```

Use: calculate the sum of elements.

Parameter Description:

Value: Double type or bigint type. If the input is ‘string’ type, it will be converted to ‘double’ type by implicit conversion and then involved in operation. If the value in column is NULL, this row will not be involved in calculation. Boolean type is not allowed involving in calculation.

Return value: if the input parameter is ‘bigint’ type, return ‘bigint’ type. If the input parameter is ‘double’ type or ‘string’ type, return ‘double’ type.

Example:

```
--If the table ‘tbla’ has a column ‘value’ and its data type is Bigint.

+------+
| value|
+------+
| 1      |
+------+
| 2      |
+------+
| NULL |
+------+
select sum(value) from tbla; --return value is 3.
```

WM_CONCAT

Function Definition:

```
string wm_concat(string separator, string str)
```

Use: use specified separator to link the value in 'str'.

Parameter Description:

Separator: a String type constant, the separator. If it is other data type or is not a constant, expection will be caused.

'str': String type. If the input is 'bigint' type or 'double' type or 'datetime' type, it will be converted to 'string' type by implicit conversion and then involved in operation. If it is other data type, expection will be reported.

Return value: String type.

Note:

- For the sentence "select wm_concat (',', name) from test_src;", if 'test_src' is empty set, this ODPS SQL sentence will return NULL.

3.3.3.6 Other Functions

CAST

Function Definition:

```
cast(expr as <type>)
```

Use: convert the result of expression to object type. For example, cast ('1' as bigint) is to convert string '1' to bingint '1'. If the conversion is unsuccessful or the conversion is not supported, expection will be caused.

Note:

- cast (double as bigint): convert double type value to bigint type value.
- cast (string as bigint): while the string is converted to 'bigint' type, if the digits indicated by 'int' exist in this string, it will be converted to 'bigint' type directly. If the digits indicated by 'float' or 'exponent' exist in this string, it will be converted to 'double' type at first and then converted to 'bigint' type.
- For cast (string as datetime) or cast (datetime as string), it will adopt default

datatime format yyyy-mm-dd hh: mi: ss.

COALESCE

Function Definition:

```
coalesce(expr1, expr2, ...)
```

Use: return the first value which is not NULL from the list. If all values in the list are NULL, return NULL.

Parameter Description:

Expri: value to be tested. All these values should have the same data type or be NULL, otherwise expction will be caused.

Return value: return value type is the same as parameter type.

🕮 **Note:**

There must be one parameter at least, otherwise expection will be caused.

DECODE

Function Definition:

```
decode(expression, search, result[, search, result]...[, default])
```

Use: Implement the selection function of If-Then-Else branch.

Parameter Description:

Expression: expression to be compared.

Search: the search item; needs to be compared with ‘expression’.

Result: return value afther ‘search’ matches ‘expression’.

Default: it is an option. If all search items do not match the expression, return this default value. If it is not specified, return NULL.

Return value: return matched ‘search’. If there is no matching record, return ‘default’. If ‘default’ is not specified, return NULL.

🕮 **Note:**

- You must specify at least three parameters.
- All of the result types must be the same or NULL. Inconsistent data type will cause an exception. All of the ‘search’ and ‘expression’ types must be consistent, otherwise expection will be reported.
- If the option ‘search’ in ‘decode’ has repeated record and has been matched, return the first value.

Example:

```
Select
   decode(customer_id,
            1, 'Taobao',
            2, 'Alipay',
            3, 'Aliyun',
            NULL, 'N/A',
            'Others') as result
from sale_detail;
```

The 'decode' function mentioned above implements the function in following 'if-then-else' sentence:

```
if customer_id = 1 then
    result := 'Taobao';
elsif customer_id = 2 then
    result := 'Alipay';
elsif customer_id = 3 then
    result := 'Aliyun';
...
else
    result := 'Others';
end if;
```

But you need to note that calculating 'NULL= NULL' by ODPS SQL, return NULL, while the values of NULL and NULL are equal in decode function. In the example mentioned above, when the vaue of 'customer_id' is NULL, 'decode' function returns 'N/A' as a result.

GREATEST

Function Definition:

```
greatest(var1, var2, …)
```

Use: return the greatest input parameter.

Parameter Description:

var1, var2 can be 'bigint' type or 'double' type or 'datetime' type or 'string' type. If all values are NULL, return NULL.

Return value: The greatest value in input parameter. If the implicit conversion is not needed, return type is the same as input parameter type.

NULL is the least value.

If the input parameter types are different, for 'double' type, 'bigint' type and 'string' type, convert them to be 'double' type. For 'string' type and 'datetime' type, convert them to be 'datetime' type. Other implicit conversion is not allowed.

ORDINAL

Function Definition:

```
ordinal(bigint nth, var1, var2, …)
```

Use: return the location value specified by ‘nth’ after the input variables are sorted by small to large.

Parameter Description:

‘nth’: Bigint type, specify the location to return its value. If it is NULL, return NULL.

‘var1’, ‘var2’: its type can be ‘bigint’ or ‘double’ or ‘datetime’ or ‘string’.

Return value: The value in nth bit. If the implicit conversion is not needed, return type is the same as input parameter type.

If there is implicit conversion in input parameters, for ‘double’ type, ‘bigint’ type and ‘string’ type, convert them to be ‘double’ type. For ‘string’ type and ‘datetime’ type, convert them to be ‘datetime’ type. Other implicit conversion is not allowed.

NULL is the least value.

Example:

```
ordinal(3, 1, 3, 2, 5, 2, 4, 6) = 2
```

LEAST

Function Definition:

```
least(var1, var2, …)
```

Use: return the least value in input parameter.

Parameter Description:

‘var1’/‘var2’: Its type can be ‘bigint’ or ‘double’ or ‘datetime’ or ‘string’. If all values are NULL, return NULL.

Return value:

The least value in input parameter; If the implicit conversion is not needed, return type is the same as input parameter type.

NULL is the least value.

If there is implicit conversion in input parameters, for ‘double’ type, ‘bigint’ type and ‘string’ type, convert them to be ‘double’ type. For ‘string’ type and ‘datetime’ type, convert them to be ‘datetime’ type. Other implicit conversion is not allowed.

UUID

Function Definition:

```
string uuid()
```

Use: return a random ID.

Example: “29347a88-1e57-41ae-bb68-a9edbdd94212”.

SAMPLE

Function Definition:

```
boolean sample(x, y, column_name)
```

Use: sample all values of ‘column_name’ according to the setting of x and y and filter out the rows which do not meet the sampling condition.

Parameter Description:

‘x’, ‘y’: Bigint type, indicates ‘hash’ to x portions, take yth portions. ‘y’ can be ignored. If ‘y’ is ignored, take the first portion. If ‘y’ in parameter is ignored, then ‘column_name’ will be ignored at the same time. ‘x’ and ‘y’ are bigint constants and greater than 0. If it is other data type or less than or equal to 0, expection will be thrown. If y>x, expection will also be thrown. If any input of ‘x’ and ‘y’ is NULL, return NULL.

column_name: object column of sampling. ‘column_name’ can be ignored. If it is ignored, do random sampling according to x and y. It can be any data type and the column value can be NULL. Do not need implicit type conversion. If ‘column_name’ is the constant NULL, expection will be reported.

Return value: Boolean type.

Note:

- In order to avoid data skew brought by NULL value, so NULL values in column_name will be carried out a uniform hash in x portions. If ‘column_name’ is not added, the output is not necessarily uniform since the data size is smaller. So ‘column_name’ is suggested to be added in order to get better output.

Example:

Suppose that the table ‘tbla’ is existent and there is a column ‘cola’ in this table:

```
select * from tbla where sample (4, 1 , cola) = true;
--The values will be carried out Hash into 4 portions and take the first portion.
select * from tbla where sample (4, 2) = true;
--The values will do random Hash into 4 portions for each row of data and take the second portion.
```

3.3.4 UDF

3.3.4.1 Summary

The full name of UDF is User Defined Function. ODPS provides many built-in functions to meet user’s computing requests and user can also create user-defined functions to meet different computing needs. UDF in use is similar with ordinary Bulit-in Function. At present, SQL UDF function is not opened. If you need to use this function, you can get an invitation code by Aliyun official website.

In ODPS, user can expand two kinds of UDF:

UDF Class	Description
User Defined Scalar Function (also called UDF).	User Defined Scalar Function also called UDF. The relationship between input and output is one-to-one relationship. Read a row data and write an output value.
UDTF(User Defined Table Valued Function)	User Defined Table Valued Function, used to solve the scene that calling one function is to output multiple rows of data. It is a unique defined function which can return multiple fields, while UDF only outputs a return value.
UDAF(User Defined Aggregation Function)	User Defined Aggregation Function, the relationship between its input and output is one-to-many relationship. That is to aggregate multiple input records to an output value. It can be used with 'Group By' clause together.

🕮 **Note:**

- Broadly speaking, 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 narrow sense, it just represents user Defined Scalar Function. The document will use this term frequently and the readers should judge the specific meaning according to the context.

3.3.4.1.1 Parameter Type and Return Value Type

The data types of UDF supported by ODPS SQL include: bigint, string, double, Boolean and datetime. The correspondence of ODPS data types and Java types is shown as follows:

ODPS SQL Type	Bigint	String	Double	Boolean	Datetime
Java Type	Long	String	Double	Boolean	time

🕮 **Note:**

- NULL value in SQL is represented by NULL value in Java, so it is not allowed to use 'Java primitive type', which can not represent NULL value in SQL.
- User is not suggested to use Double.nan, which is not defined in ODPS.

3.3.4.2 UDF

To implement UDF, the class 'com.aliyun.odps.udf.UDF' should be inherited and the 'evaluate' method shoule 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 will match correct evaluate method according to the parameter type called by UDF.

Next gives a UDF example:

```
package org.alidata.odps.udf.examples;
```

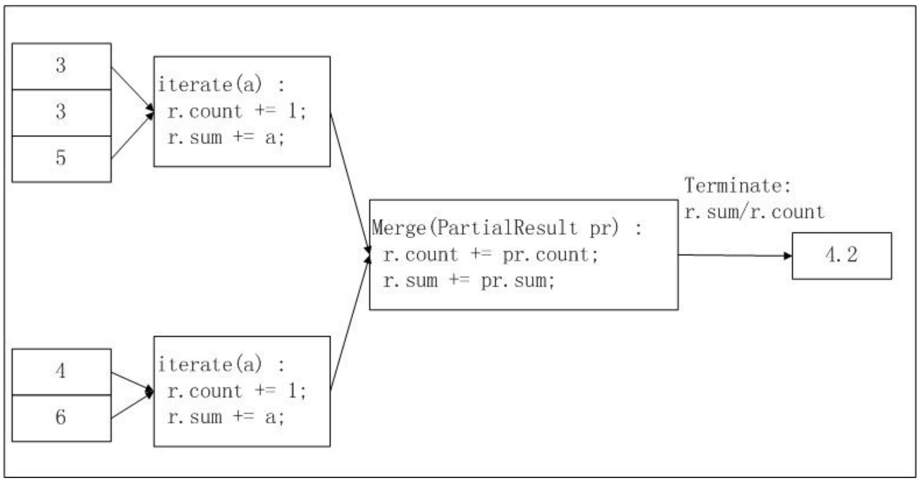
```
import com.aliyun.odps.udf.UDF;


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

3.3.4.3 UDAF

To implement Java UDAF, you need to inherit the class 'com.aliyun.odps.udf.UDAF' and the following interfaces will 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;
}
```

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 ODPS UDAF:



In the figure displayed above, the input data is sliced according to certain size. The size of each slice is suitable for a worker completed in appropriate time. This slice size needs to be condigured by user manually. The calculation process of UDAF is divided into two stages:

In the first stage, each worker will count the data quantity and total sum in slice. You can take the data quantity and total sum in each slice as an intermediate result;

In the second stage, worker will gather the information of each slice generated in first stage. In the final output, r.sum / r.count is the average of all input data.

Next 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;
  }

}
```

3.3.4.4 UDTF

3.3.4.4.1 Summary

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	Initialization method. Before UDTF processes the input data, it will transfer the user-fefined initialization behavior. In every Worker, ‘setup’ will be called once at first.
public void process(Object[] args) throws UDFException	This method is called by framework. Each record in SQL will correspondingly call process once. The parameter of ‘process’ is UDTF input parameter specified in SQL. The input parameter is input in form of ‘Object []’ and the result is output through the function ‘forward’. User needs to call ‘forward’ by himself in process functions, to determine the output data.
public void close() throws UDFException	End method of UDTF, which is called by framework and can only be called for once after the last record has been processed.
public void forward(Object ...o) throws UDFException	User calls ‘forward’ method to outputthe data. Every ‘forward’ depends on outputting one record. It corresponds to the column specified by ‘as’ clause of UDTF 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 defines 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);
      }
   }
}
```

In SQL you can use this UDTF as following example. Suppose that the register function name in ODPS 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   |
+----+----+
```

3.3.4.4.2 UDTF Instructions for Use

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

```
select user_udtf(col0, col1, col2) as (c0, c1) from my_table;
select user_udtf(col0, col1, col2) as (c0, c1) from
      (select * from my_table distribute by key sort by key) t;
```

```
select reduce_udtf(col0, col1, col2) as (c0, c1) from
    (select col0, col1, col2 from
        (select map_udtf(a0, a1, a2, a3) as (col0, col1, col2) from my_table) t1
    distribute by col0 sort by col0, col1) t2;
```

But using UDTF has limitations:

Other expressions are not allowed in the same SELECT clause:

```
select value, user_udtf(key) as mycol ...
```

UDTF can not 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
```

3.3.4.4.3 Other UDTF Examples

In UDTF, you can read ODPS resources.

```
package org.alidata.odps.udtf.examples.resource;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.HashMap;

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;

@Resolve("string->string")
public class ExampleUDTF extends UDTF {

  ExecutionContext ctx;
  HashMap<String, String> replaceMap;

@Override
  public void setup(ExecutionContext ctx) throws UDFException {
    this.ctx = ctx;
    replaceMap = new HashMap<String, String>();
            try {
              BufferedInputStream                    resource                =
ctx.readResourceFileAsStream("replace_map.txt");
            BufferedReader     reader     =     new     BufferedReader(new
InputStreamReader(resource));
            String st = null;
            while ((st = reader.readLine()) != null) {
              String[] replaceStr = st.split(",");
              if (replaceStr.length != 2) {
                throw new UDFException("resource format error.");
               }
              } catch (IOException e) {
               throw new UDFException(e);
         }

      }

    @Override
    public void process(Object[] args) throws UDFException {
      String inputLine = (String) args[0];
      if (replaceMap.containsKey(inputLine)) {
        forward(replaceMap.get(inputLine));
      } else {
        forward(inputLine);
      }
    }
  }
```

Note:

- At present, ODPS also supports reading resources from UDF. In MapReduce, you can also use the same way to access resources.

4 MapReduce

4.1 Summary

4.1.1 MapReduce

ODPS provides MapReduce programming interface. User can use Java API to write MapReduce program for processing data in ODPS. At present, the MapReduce function is not opened. If you want to use this function, you should apply for an invitation code from Aliyun website. This section only introduces how to use MapReduce SDK simply. For a detailed description of the MapReduce SDK, please refer to official Java Doc.

MapReduce is a distributed data processing model proposed by Google at first. It has drawn a lot of attention and has been applied to all kinds of business scenarios. The data processing process of MapReduce is mainly divided into two stages: Map stage and Reduce stage. Map stage is executed before Reduce stage. The processing logic of Map and Reduce is defined by user, but should comply with the MapReduce framework protocol.

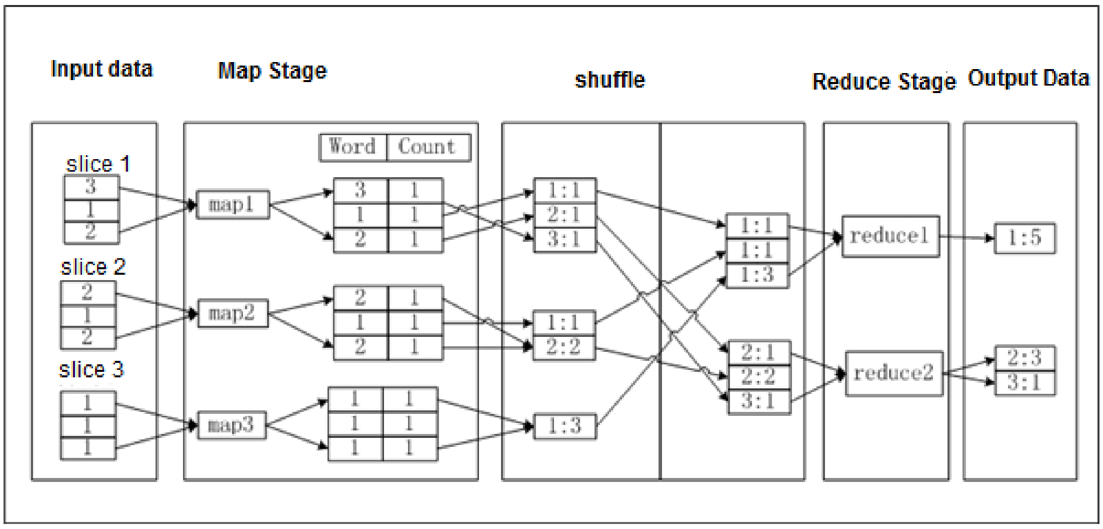
Before executing Map, the input data must be sliced. 'Slicing', is to divide input data into blocks of equal size. Each block is processed as the input of a single Map Worker, so that multiple Map Workers can work simultaneously.

After a slice is finished, multiple Map Workers can work simultaneously. Each Map Worker will do computing after reading their data and output the result to Reduce. While Map Worker outputs the data, it needs to specify a key for each output record. The value of this Key determines which Reduce Worker this data will be sent to. The relationship between key value and Reduce Worker is many-to-one relationship. Data with the same key will be sent to the same Reduce Worker. A single Reduce Worker may receive the data with multiple key values.

Before Reduce stage, MapReduce framework will sort the data according to their Key values, and make sure data with same Key value will be grouped together. If user specifies 'Combiner', the framework will call Combiner to aggregate the same key data. The logic of Combiner is also defined by user. Differently from the classical MapReduce framework, the input parameter and output parameter of Combiner must be consistent with Reduce in ODPS. This processing is usually called 'Shuffle'.

Next stage is Reduce. Data with the same key will be shuffled to the same Reduce Worker. A Reduce Worker will receive data from multiple Map Workers. Each Reduce Worker will execute Reduce operation for multiple records of the same key. At last, multiple records of same key will become a value through Reduce processing.

The following will take WordCount as an example, to explain the concepts of ODPS MapReduce stages. Suppose there is a text 'a.txt', where each row is indicted by a number. We need to count the appearance occurrence of each number. The number in the text is called 'Word' and the number appearance occurrence is called Count. To complete this function by ODPS Mapreduce, we will go through the steps described in following figure:



At first, slice the text and take the data in each slice as the input of single Map Worker.

Map processes the input. Once getting a number, set Count to 1. Then output the <Word, Count> queue. Now take 'Word' as the Key of output data.

In the prophase of Shuffle stage, sort the output of each Map Worker according to Key value (value of Word) at first. Then execute Combine operation after sorting, that is to accumulate the Count of same Key value (Word value) to constitute a new <Word, Count> queue. This process is called combiner sorting.

In the later stage of Shuffle, data is transmitted to Reduce. Reduce Worker will sort the data based on Key value again after receiving data.

At the time of processing data, each Reduce Worker adopts same logic with Combiner by accumulating Count with same Key value (Word value) to get the output result.

Note:

As data in ODPS are stored in tables, the input and output of ODPS MapReduce can only be a table. User-defined output is not allowed and the similar file system interface is not provided.

4.1.2 Extensional MapReduce

The traditional MapReduce model requires that the data must be loaded to the distributed file system (such as HDFS or ODPS table) after each round of MapReduce operation. But the general MapReduce application usually consists of multiple MapReduce jobs and each job output needs to be written to disk in the end. The following Map task is only to read data, prepared for subsequent Shuffle stage.That actually caused redundant IO operations.

The calculation scheduling logic of ODPS can support more complex programming paradigm. In the case mentioned above, the next Reduce operation can be executed after 'Reduce' operation and inserting a Map operation is not necessary. Therefore, ODPS provides an extensional MapReduce model. That is to say, any numbers of Reduce operations can follow Map operation, such as Map-Reduce-Reduce.

Hadoop Chain Mappper/Reducer also supports analogous serial Map or Reduce operations, but has essential difference from extensional ODPS (MR2) model. Chain Mapper/Reducer is based on traditional MapReduce model and it can only add one or multiple Mapper operations (it is not allowed to add Reducer.) followed the original Mapper or Reducer. The following benefits 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 IO model are not changed essentially.

4.2 Introduction

4.2.1 Running Commands

ODPS client provides a jar command to run MapReduce job. The detailed syntax is shown as follows:

```
Usage: jar [<GENERIC_OPTIONS>] <MAIN_CLASS> [ARGS]
    -conf <configuration_file>   Specify an application configuration file
    -classpath <local_file_list>   classpaths used to run mainClass
    -D <name>=<value>       Property value pair, which will be used to run mainClass
    -local       Run job in local mode
    -resources <resource_name_list>     file/table resources used in mapper or reducer,
seperate by comma
```

<GENERIC_OPTIONS> includes the following parameters (optional parameters):

-conf <configuration file>: speficy the configuration file

-classpath <local_file_list>: the classpath used to specify the jar package of function 'main'. In most cases, users are more accustomed to write the main function and Map/Reduce function in a package. For example: WordCount Example. Therefore, in the running period of the example program, mapreduce-examples.jar appears in '-resources' parameter and '-classpath' parameter, both have different meaning. '-resources' quotes Map/Reduce function, running in distributed environment while '-classpath' quotes 'Main' function, running on the local. The specified path of jar package is also a local path. The package name is separated by system default file separator. (For Windows system, use a semicolon ';', while for Linux system, use a colon ':').

-D <prop_name>=<prop_value>: Multiple Java properties of <mainClass> in local mode can be defined.

-local: execute MapReduce job in local mode, mainly used for program debugging.

-resources <resource_name_list>: the resource declaration used in MapReduce running time; generally, the resource name in which Map/Reduce function is included should be specified in 'resource_name_list'. If the user has read other ODPS resources in the Map/Reduce function, then these resource names also need to be added in 'source_name_list'. The resources are separated by commas. If you need use span project resources, you should add the prefix 'PROJECT/resources/'. For example: -resources otherproject/resources/resfile. About the example how to read the resource in the Map/Reduce function, please refer to Use Resource Example.

User can specify the configuration file 'JobConf' by option '-conf'. This file can influence JobConf settings in SDK. Next gives an example of configuration file 'JobConf':

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

In the example mentioned above, a variable named 'import.filename' is defined and its value is 'resource.txt'. User can get this variable value through JobConf interface in MapReduce program. User can achieve the same purpose through JobConf interface in SDK. For more on detailed useage method, please refer to Use

Resource Example.

Example:

```
jar -resources mapreduce-examples.jar -classpath mapreduce-examples.jar
    org.alidata.odps.mr.examples.WordCount wc_in wc_out

create resource file data/src.txt
jar -resources src.txt,mapreduce-examples.jar -classpath mapreduce-examples.jar
    org.alidata.odps.mr.examples.WordCount wc_in wc_out

create resource file data/a.txt
create resource table wc_in as test_table
create resource jar work.jar
jar -conf odps-mapred.xml -resources a.txt,test_table,work.jar
    -classpath work.jar:otherlib.jar
-D import.filename=resource.txt org.alidata.odps.mr.examples.WordCount args ...
```

4.2.2 Input and Output

The data types of input and output just support ODPS Built In data types: Bigint, Double, String and Boolean. User-defined types and ‘Datetime’ type are not supported.

Multiple-table input is acceptable and the schema of input tables can be different. In Map function, user can obtain corresponding Table information of present 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; the default output doe not need lablel. But at present the situation that there is no output is not acceptable.

4.2.3 Function and Methods

Map/Reduce

Map and Reduce support corresponding map/reduce method, setup method and cleanup method respectively. The setup method is called before the map/reduce method; each worker will call it and can only call once. The reduce method is called after the map/reduce method; each work will call it and only call once.

Sort/Group

A few columns in output key records can be taken as sort columns, but user-defined comparator is not supported. User can select several columns from sort columns as

Group columns, but user defined Group comparator is not supported. Sort columns are used to sort user data while Group columns are used for secondarySort.

Partitioner

To set partition column and user defined function (partitioner) is supported. The use priority of partition column is higher than partitioner. The partitioner is used to distribute the output data on Map terminal to different Reduce Workers according to Hash logic.

Combiner

The function Combiner is used to combine adjacent records in Shuffle stage. User can choose whether to use Combiner according to different business logic. Combiner is a kind of optimization of MapReduce computing framework and the logic of Combiner is usually the same as Reduce. After Map outputs the data, the framework will do local combiner operation for the data which has the same key value on Map terminal.

Read Resource

User is allowed to read ODPS resources in map/reduce. Any Worker of map/reduce will load the resources to memory for user code use.

4.2.4 Local Operation

By setting '-local' parameter in jar command, user can simulate MapReduce running process on the local to carry on local debugging. At local operation time, the client will download required Meta information of input tables, resources and Meta information of output tables from ODPS and save them into a local directory named 'warehouse'.After the program running ends, the calculation result will be 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 will be referenced directly at next running time and will not repeat the downloading.

In the local operation course, multiple Map and Reduce workers will still be 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:

There is limitation for row number of input table: at present, at most 100 rows of data can be downloaded.

Usage of resource: in distributed environment, ODPS will limit the size of referenced resource. About the detail, please refer to Application Limitations. But in local running environment, there is no limitation for resource size.

Security limitation: ODPS MapReduce and UDF program running in a distributed environment will be limited by Java SDK. But there is no this limitation in local operation

A local operation example is shown as follows:

```
odps:my_project> jar -local 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
```

Note:

- About the WordCount example, please refer to WordCount Example.
- If user runs local debugging command for the first time, a path named 'warehouse' will appear in current path after the command is executed successfully. The directory structure of warehouse is shown as follows:

```
warehouse
    |______my_project (project directory)
            |______wc_in (table directory)
            |            |____ data (file)
            |            |____ __schema__ (file)
            |
            |______wc_in (table directory)
                        |____ data (file)
                        |____ __schema__ (file)
```

The same level directory of my_project indicates the project. 'wc_in' and 'wc_out' indicate tables. The table files read by user in jar command will be downloaded into this directory. The contents in' __schema' indicate table Meta information, which 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
```

Among the format, column and column type are separated by colon ':' and column and column are separated by comma ‘,’. In the front of schema file needs to declare the Project name and Table name, like ‘projectname.tablename’, separated by comma and column definition. The file ‘data; indicates table data. The column quantity and corresponding data should comply with the definition in ‘__schema’. The extra column and missing column are not allowed. The column and another column are separated by comma ‘,’.

The contents of ‘_schema’ in wc_in:

```
my_project.wc_in,key:STRING,value:STRING
```

The content of ‘data’:

```
0,2
```

The client will download the Meta information of table and part of data from ODPS and save them into two files mentioned above. If you need run this example again, the data in the directory ‘wc_in’ will be used directly and will not be downloaded again. Here need a special statement: the function to download data from ODPS is only supported in MapReduce local operation mode. If the local debugging is executed in Eclipse development plug-in, the data of ODPS cannot be downloaded to local.

The contents of ‘_schema’ in wc_out:

```
my_project.wc_out,key:STRING,cnt:BIGINT
```

The content of ‘data’:

```
0,1
2,1
```

The client will download the Meta information of wc_out from ODPS and save it to the file ‘_schema’. The file ‘data’ is a result data file generated after local operation.

📖 **Note:**

- User can edit ‘_schema’ and ‘data’ and put these two files into corresponding table directory. When running on the local, the client detects the table directory already exists and will not download the information of this table from ODPS. The table directory on the local can be a table that does not exist in ODPS.

4.2.5 Application Limitations

At present, the application limitations of ODPS incude:

The length of string column in ODPS table can not exceed 2MB.

The quantity of resources referenced by single task can not exceed 512 and the partition table is calculated as one unit.

The total size of resources referenced by single task can not exceed 64MB.

The inputs and outputs of single task can not exceed 128 respectively.

The quantity of Counter defined in single task can not exceed 64.

The size of memory occupied by single Map or Reduce worker defaults to 2048 and the rangle is [256MB, 12GB].

The times of single Map or Reduce Worker reading a resource limit to <=64.

In local running mode, the number of Map Worker and Reduce worker can not exceed 100. The default downloaded record number of one input is 100.

4.3 Program Samples

4.3.1 WordCount Example

Sample code (for reference only):

```
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 Example

For the MapOnly job, Map outputs the information of < Key, Value > to the table in ODPS directly. User only needs to specify the output table and does not need to specify the key/value Meta information output by Map.

Sample code (for reference only):

```
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 Multiple Inputs and Outputs Example

At present, ODPS supports the input and output of multiple tables. In multiple inputs, the column quantity and data types of multiple tables must be the same. IN multiple outputs, the column quantity and data types can be different.

Sample code (for reference only):

```
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.
 *
 * To run: jar -resources odps-mapred-example-0.12.0.jar
com.aliyun.odps.mapred.open.example.MultipleInOut
* mr_src,mr_src1,mr_srcpart|pt=1,mr_srcpart|pt=2/ds=2
*
mr_multiinout_out1,mr_multiinout_out2|a=1/b=1|out1,mr_multiinout_out2|a=2/b=2|out2;
 *
 **/
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);
                //if not specify label, output default outputs.
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, "out1");
          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"));

        //analyze the string in user input table.
        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);
          }
        }

        // analyze the string in user output table.
        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 Multiple Tasks Example

Sample code (for reference only):

```
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
 *
 * To run: jar -resources multijobs_res_table,odps-mapred-example-0.12.0.jar
 * com.aliyun.odps.mapred.open.example.MultiJobs mr_multijobs_out;
 *
 **/
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 {
      //Get the variable defined in Main 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 SecondarySort Example

Sample code (for reference only):

```
      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.
 *
 * To run: jar -resources odps-mapred-example-0.12.0.jar
 * com.aliyun.odps.mapred.open.example.SecondarySort mr_sort_in
mr_secondarysort_out;
 *
 **/
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 be 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.setMapOutputValueSchema(SchemaUtils.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 Use Resource Example

Sample code (for reference only):

```
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
  *
  * To run: jar -resources odps-mapred-example-0.12.0.jar,mr_join_src1.txt
  * com.aliyun.odps.mapred.open.example.Upload mr_join_src1.txt mr_join_src1;
  *
  **/
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 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. About the use method of '–conf', please refer to Running Commands.

4.3.7 Use Counter Example

The code example defines three Counters: map_outputs, reduce_outputs and global_counts. User can get any user-defined Counter from setup interface, map/reduce itterface and cleanup interface and then execute operation.

Sample code (for reference only):

```
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
     *
     * To run: jar -resources odps-mapred-example-0.12.0.jar
     * com.aliyun.odps.mapred.open.example.UserDefinedCounters mr_src
mr_testcounters_out;
     *
     **/
    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 total_tasks = context.getCounter(MyCounter.TOTAL_TASKS);
                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 Example

Sample code (for reference only):

```
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.
 *
 * To run: jar -resources odps-mapred-example-0.12.0.jar
 * com.aliyun.odps.mapred.open.example.Grep mr_src mr_grep_tmp mr_grep_out
val;
 **/
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 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" }); // sort by
 // decreasing
// freq
```

```
        @SuppressWarnings("unused")
        RunningJob rjSort = JobClient.runJob(sortJob);
    }


}
```

4.3.9 Join Example

ODPS MapReduce framework does not support Join. But you can implement data join in your Map/Reduce function.

Suppose that you need to join two tables ‘mr_join_src1(key bigint, value string)’ and ‘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.

Sample code (for reference only):

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


    import java.io.IOException;
    import java.util.ArrayList;
    import java.util.Iterator;
    import java.util.List;



    import com.aliyun.odps.counter.Counter;
    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;
    /**
    * Join
    *
    * To run: jar -resources odps-mapred-example-0.12.0.jar
    * com.aliyun.odps.mapred.open.example.Join mr_join_src11 mr_join_src22
mr_join_out;
    *
    **/
    public class Join {

     public static class JoinMapper extends MapperBase {
        private Record mapkey;
        private Record mapvalue;

        @Override
        public void setup(TaskContext context) throws IOException {
          mapkey = context.createMapOutputKeyRecord();
          mapvalue = context.createMapOutputValueRecord();
        }

        @Override
        public void map(long key, Record record, TaskContext context)
            throws IOException {
/*Judge which table this record comes from according to 'value'. Here belongs to user
code logic.
If you can not judge which table the field belongs to through the 'value', you can consider
add a field in the input table.
The tag generated by 'value' will be used in join operation in Reduce. */
          long tag = 1;
          String val = record.get(1).toString();
          if (val.startsWith("valb_")) {
            tag = 2;
          }
          mapkey.set(0, Long.parseLong(record.get(0).toString()));
          mapkey.set(1, tag);

          mapvalue.set(0, tag);
          for (int i = 1; i < record.getColumnCount(); i++) {
            mapvalue.set(i, 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 = (Long) key.get(0);
          List<Object[]> list1 = new ArrayList<Object[]>();

          Counter cnt = context.getCounter("MyCounters", "reduce_outputs");
          cnt.increment(1);
          while (values.hasNext()) {
            Record value = values.next();
            long tag = (Long) value.get(0);
            if (tag == 1) {
//if the data is from the first table, cache the data into list.
//Because the data is sorted according to Key and tag, the values (tag==1) will be sorted in
front.
//In actual computing, user is suggested to use this method cautiously. If the values of a
certain key are too much, it will cause the memory increasing of Reduce. If the memory
size exceeds the set value of ‘JobConf::setMemoryForReduceTask’, Reduce may be killed
by system.
                    list1.add (value.toArray ().clone ());
                } else {
//If the data is from the second table, do sorting according to Key and tag. Now all values
of the first table have been saved in list1.
//For all values in the first table:
                    for (Object r1: list1) {
                        int index = 0;
                        //At first, set Key
                        result.set(index++, k);
                        Object[] s_arr = (Object[])r1;
                        result.set(index++, s_arr[1].toString());
                        result.set(index++, value.get(1).toString());
                        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("tagx:bigint,value:string"));

        job.setPartitionColumns(new String[] { "key" });
//use key and tag to do sorting. Only after the key has been sorted, join operation can be
executed in Reduce terminal.
//Tag is used to distinguish which table the current Record comes from.
        job.setOutputKeySortColumns(new String[] { "key", "tag" });
        job.setOutputGroupingColumns(new String[] { "key" });

 //As Reduce terminal uses list to cache data, to increase the memory of Reduce Worker
is suggested.
        job.setMemoryForReduceTask(4096);
        job.setInt("table.counter", 0);

        InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(), job);
        InputUtils.addTable(TableInfo.builder().tableName(args[1]).build(), job);
        OutputUtils.addTable(TableInfo.builder().tableName(args[2]).build(), job);

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

4.3.10 Sleep Example

Code example, only for reference:

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


    import java.io.IOException;
    import java.util.Iterator;
```

```
import java.util.LinkedHashMap;
import java.util.Map;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.aliyun.odps.OdpsException;
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;

/**
 * Dummy class for testing MR framefork. Sleeps for a defined period of time in
 * mapper and reducer. Generates fake input for map / reduce jobs. Note that
 * generated number of input pairs is in the order of
 * <code>numMappers * mapSleepTime / 100</code>, so the job uses some disk
 * space.
 * To run: jar -resources odps-mapred-example-0.12.0.jar
com.aliyun.odps.mapred.open.example.SleepJob
 * -m 1 -r 1 -mt 1 -rt 1;
 *
 **/
public class SleepJob {

  private static Log LOG = LogFactory.getLog(SleepJob.class);

  public static class SleepMapper extends MapperBase {
        private LinkedHashMap<Integer, Integer> inputs = new
LinkedHashMap<Integer, Integer>();
        private long mapSleepDuration = 100;
        private int mapSleepCount = 1;
        private int count = 0;
        private Record key;

        @Override
        public void setup(TaskContext context) throws IOException {
          LOG.info("map setup called");
          JobConf conf = (JobConf) context.getJobConf();

          mapSleepCount = conf.getInt("sleep.job.map.sleep.count", 1);
          if (mapSleepCount < 0)
            throw new IOException("Invalid map count: " + mapSleepCount);

          mapSleepDuration = conf.getLong("sleep.job.map.sleep.time", 100)
              / mapSleepCount;

          LOG.info("mapSleepCount = " + mapSleepCount + ", mapSleepDuration = "
              + mapSleepDuration);

          final int redcount = conf.getInt("sleep.job.reduce.sleep.count", 1);
          if (redcount < 0)
            throw new IOException("Invalid reduce count: " + redcount);

          final int emitPerMapTask = (redcount * conf.getNumReduceTasks());

          int records = 0;
          int emitCount = 0;

          while (records++ < mapSleepCount) {
            int key = emitCount;
            int emit = emitPerMapTask / mapSleepCount;
            if ((emitPerMapTask) % mapSleepCount > records) {
              ++emit;
            }
            emitCount += emit;
            int value = emit;
            inputs.put(key, value);
          }

          key = context.createMapOutputKeyRecord();
        }

        @Override
        public void cleanup(TaskContext context) throws IOException {
        // it is expected that every map processes mapSleepCount number of records.
          LOG.info("map run called");

          for (Map.Entry<Integer, Integer> entry : inputs.entrySet()) {
            LOG.info("Sleeping... (" + (mapSleepDuration * (mapSleepCount - count))
                + ") ms left");
            try {
              Thread.sleep(mapSleepDuration);
            } catch (InterruptedException e) {
              throw new IOException(e);
            }

            ++count;
// output reduceSleepCount * numReduce number of random values, so that each reducer
will get reduceSleepCount number of keys.
            int k = entry.getKey();
            int v = entry.getValue();
            for (int i = 0; i < v; ++i) {
              key.set(new Object[] { (Long) ((long) (k + i)) });
              context.write(key, key);
            }
          }
        }
```

```
}

public static class SleepReducer extends ReducerBase {

  private long reduceSleepDuration = 100;

  private int reduceSleepCount = 1;
  private int count = 0;

  @Override
  public void setup(TaskContext context) throws IOException {
    LOG.info("reduce setup called");
    JobConf conf = (JobConf) context.getJobConf();
    reduceSleepCount = conf.getInt("sleep.job.reduce.sleep.count",
        reduceSleepCount);
    reduceSleepDuration = conf.getLong("sleep.job.reduce.sleep.time", 100)
        / reduceSleepCount;
    LOG.info("reduceSleepCount = " + reduceSleepCount
        + ", reduceSleepDuration = " + reduceSleepDuration);
  }

  @Override
  public void reduce(Record key, Iterator<Record> values, TaskContext context)
      throws IOException {
    LOG.info("reduce called");

    LOG.info("Sleeping... ("
        + (reduceSleepDuration * (reduceSleepCount - count)) + ") ms left");
    try {
      Thread.sleep(reduceSleepDuration);
    } catch (InterruptedException e) {
      throw new IOException(e);
    }

    count++;
```

```
        }
    }

    public static int run(int numMapper, int numReducer, long mapSleepTime,
        int mapSleepCount, long reduceSleepTime, int reduceSleepCount)
        throws OdpsException {
      JobConf job = setupJobConf(numMapper, numReducer, mapSleepTime,
          mapSleepCount, reduceSleepTime, reduceSleepCount);
      JobClient.runJob(job);
      return 0;
    }

    public static JobConf setupJobConf(int numMapper, int numReducer,
        long mapSleepTime, int mapSleepCount, long reduceSleepTime,
        int reduceSleepCount) {
      JobConf job = new JobConf();

      InputUtils.addTable(TableInfo.builder().tableName("mr_empty").build(), job);
      OutputUtils.addTable(TableInfo.builder().tableName("mr_sleep_out").build(),
job);

      job.setNumReduceTasks(numReducer);

      job.setMapperClass(SleepMapper.class);
      job.setReducerClass(SleepReducer.class);

      job.setMapOutputKeySchema(SchemaUtils.fromString("int1:bigint"));
      job.setMapOutputValueSchema(SchemaUtils.fromString("int2:bigint"));
      job.setPartitionColumns(new String[] { "int1" });

      job.setLong("sleep.job.map.sleep.time", mapSleepTime);
      job.setLong("sleep.job.reduce.sleep.time", reduceSleepTime);
      job.setInt("sleep.job.map.sleep.count", mapSleepCount);
      job.setInt("sleep.job.reduce.sleep.count", reduceSleepCount);
```

```
        return job;
    }

    private static void printUsage() {
        System.err.println("SleepJob [-m numMapper] [-r numReducer]"
                + " [-mt mapSleepTime (msec)] [-rt reduceSleepTime (msec)]"
                + " [-recordt recordSleepTime (msec)]");
    }

    public static void main(String[] args) throws Exception {

        if (args.length < 1) {
            printUsage();
            return;
        }

        int numMapper = 1, numReducer = 1;
        long mapSleepTime = 100, reduceSleepTime = 100, recSleepTime = 100;
        int mapSleepCount = 1, reduceSleepCount = 1;

        for (int i = 0; i < args.length; i++) {
            if (args[i].equals("-m")) {
                numMapper = Integer.parseInt(args[++i]);
            } else if (args[i].equals("-r")) {
                numReducer = Integer.parseInt(args[++i]);
            } else if (args[i].equals("-mt")) {
                mapSleepTime = Long.parseLong(args[++i]);
            } else if (args[i].equals("-rt")) {
                reduceSleepTime = Long.parseLong(args[++i]);
            } else if (args[i].equals("-recordt")) {
                recSleepTime = Long.parseLong(args[++i]);
            }
        }

  // sleep for *SleepTime duration in Task by recSleepTime per record
```

```
            mapSleepCount = (int) Math.ceil(mapSleepTime / ((double) recSleepTime));
            reduceSleepCount = (int) Math.ceil(reduceSleepTime
                  / ((double) recSleepTime));

            run(numMapper, numReducer, mapSleepTime, mapSleepCount,
reduceSleepTime,
                  reduceSleepCount);
        }

    }
```

4.3.11 Unique Example

Sample code (for reference only):

```
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
 *
 * To run: jar -resources odps-mapred-example-0.12.0.jar
 * com.aliyun.odps.open.mapred.example.Unique mr_sort_in mr_unique_out
 * key|value|all;
 *
 **/
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 Example

Sample code (for reference only):

```
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.
 * <p>
 * To run: jar -resources odps-mapred-example-0.12.0.jar
com.aliyun.odps.mapred.open.example.Sort
 * mr_sort_in mr_sort_out;
 *
 **/
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.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 Example

Code example 1, an example to take Partition as input and output (just for reference):

```
  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).buil
d(), job);

        JobClient.runJob(job);
      }
```

Sample code 2 (for reference only):

```
    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:**

- This is an example to combine ODPS SDK and MapReduce SDK to achieve MapReduce task. This code can not be compiled, only gives the example of main function. The ‘Applicable’ function is user logic, used to determine whether the Partition can be used as the input of MapReduce job.

4.3.14 Pipeline Example

The code example, only for reference:

```
package com.aliyun.odps.mapred.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 Histogram {

  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 num;
    private Record result;

    @Override
    public void setup(TaskContext context) throws IOException {
      num = context.createOutputKeyRecord();
      result = 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);
      }
      result.set(0, key.get(0));
      num.set(0, count);
      context.write(num, result);
    }
  }

  public static class IdentityReducer extends ReducerBase {

    @Override
    public void reduce(Record key, Iterator<Record> values, TaskContext context)
```

```
                throws IOException {
            while (values.hasNext()) {
                context.write(values.next());
            }
        }
    }

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

        Job job = new Job();



        /***
        * In the Pipeline construction process, if not specify OutputKeySortColumns，
PartitionColumns and OutputGroupingColumns of Mapper, the framework will use
OutputKey as the default configurations.
         ***/
        Pipeline pipeline = Pipeline.builder()
                .addMapper(TokenizerMapper.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("count", OdpsType.BIGINT) })
                .setOutputValueSchema(
                new Column[] { new Column("word", OdpsType.STRING)})

            .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);
    }

}
```

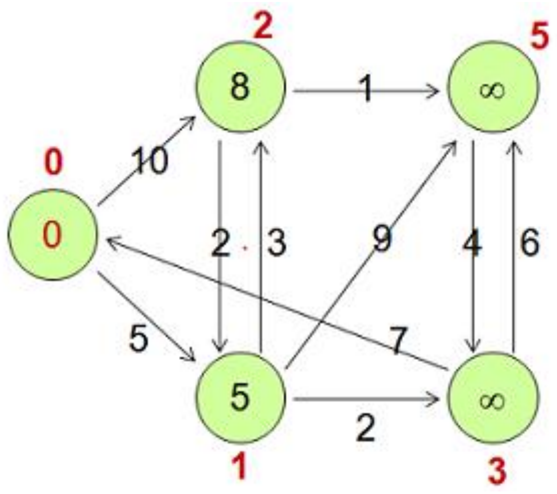
5 Graph

5.1 Summary

5.1.1 Graph Data Structure

The graph processed by ODPS Graph module must be a directed graph, composed of vertices and edges. As ODPS can only provide two-dimensional table storage structure, it requires users to map graph data into a two-dimensional table and save it in ODPS. During graph calculation and analysis stage, user defined Graphloader is used to convert two-demensional table data into vertices and edges through ODPS Graph engine. As for how to map graph data into a two-dimensional table, users can make decisions according to their own business scenarios. In Example Programs, the given examples use different table formats to express the graph data structure, for your reference.

The Vertex structure can be expressed simply as <ID, Value, Halted, Edges>, which indicates the vertex identifier (ID), weight value (Value), state (Halted, which indicates whether to stop iteration) and edge set (Edges, all edge lists which take this vertex as beginning) seperately. The Edge structure can be expressed simply as <DestVertexID, Value>, which indicates destination vertex (DestVertexID) and weight value (Value).



For example, the graph shown above is composed of the following vertices:

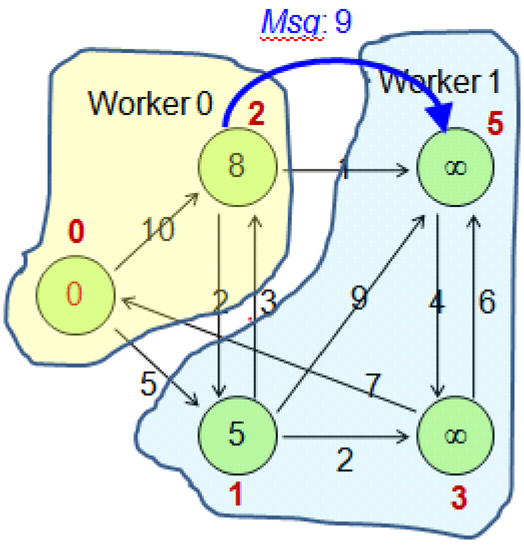
Vertex	<ID, Value, Halted, Edges>
v0	<0, 0, false, [<1, 5 >, <2, 10 >] >

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 >]>

5.1.2 Graph Programming Logic

1) Load Graph

Load graph: the framework calls user defined GraphLoader to parse input table records to vertex or edge. Distributed optimization: the framework calls user defined Partitioner to slice vertices (the default slicing logic: convert vertex ID into hash value and get modulus by Worker number) and distributes them to corresponding Workers.



For example, suppose that the worker number is 2 in the graph above, then v0 and v2 will be distributed to Worker0 because the result of ID getting modulus by 2 is 0. V1, v3 and v5 will be distributed to Worker1, because the result of ID acquiring modulus from 2 is 1.

2) Iterative calculation:

- Iteration is a ‘SuperStep’, to traverse all vertices in non-end state (Halted value is false) or the vertices which have received messages (the vertex with end state will be awaken automatically after receiving messages.) and call the method ‘compute (ComputeContext context, Iterable messages)’.
- In the method ‘compute (ComputeContext context, Iterable messages)’:

- Process the messages sent by last SuperStep to current vertex;
- Edit the graph according to actual requirements: 1) modify the value of vertex/edge; 2) send messages to certain vertices; 3) add/delete vertex or edge.
- Aggregate information to global information through Aggregator.
- Set the state of current vertex: end state or non-end state (Halted value is true or false);
- During interation processes, the framework will send messages to corresponding Worker asynchronously and process them in the next SuperStep. Users do not need to care about it.

3) Iteration termination (only need to meet any one in the following items):

- All vertices are in end state (Halted value is true) and new message is not generated;
- Reach maximum number of iterations;
- The terminate method of a certain Aggregator returns true.

Pseudo code description is shown 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();
```

5.2 Functions Description

5.2.1 Run Job

ODPS client provides a Jar command to run ODPS GRAPH jobs. Its use method is the same as Jar command in MapReduce.

```
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 will be used to run mainClass
       -local                          --Run job in local mode
       -resources <resource_name_list>    --file/table resources used in graph, seperated by
comma
```

Where <GENERIC_OPTIONS> includes (all are optional parameters):

'-conf <configuration file>': specify JobConf configuration file;

'-classpath <local_file_list &gt': classpath of local operation, used to specify the jar package which contains main function. In most cases, users are more accustomed to write main class and Graph jobs in one package, such as Single source shortest distance algorithm. Therefore in the running period of example programs, the Jar packages appear in '-resources' parameter and '-classpath' parameter, but both have different meaning. '-resources' quotes Graph job, running in distributed environment while '-classpath' quotes 'Main' function, running in the local and the specified path of jar package is also a local file path. The package name is separated by system default file separator. (For Windows system, use a semicolon ';', while for Linux system, use a colon ':').

'-D <prop_name>' =<prop_value>: Java properties of <mainClass> in local operation mode, multiple properties can be defined.

'-local': execute Graph job in local mode, mainly used for program debugging.

'-resources <resource_name_list>': the resource declaration, to be used in Graph job running time. Generally, the resource name in which Graph job is included should be specified in 'resource_name_list'. If the user has read other ODPS resources in Graph job, then these resource names also need to be added in 'source_name_list'. The resources are separated by commas. If you need use span project resources, you should add the prefix ‘PROJECT/resources/’. For example: -resources otherproject/resources/resfile.

Nevertheless, user can also run the main function of Graph job and submit the job to ODPS directly, rather than submitting job through ODPS client. Take PageRank algorithm as an example:

```
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());

  // add resources used in job to cache resource, corresponding to specified resource
following –resurces and –libjars in jar command.
job.addCacheResource("mapreduce-examples.jar");
  // Add used jar and other files into class cache resource, corresponding to specified
resource following –libjars in jar command.
  job.addCacheResourceToClassPath("mapreduce-examples.jar");
  //Set corresponding configuration items in odps_config.ini in console.
  OdpsConf.getInstance().setProjName("project_name");
  OdpsConf.getInstance().setEndpoint("end_point");
  OdpsConf.getInstance().setAccessId("access_id");
  OdpsConf.getInstance().setAccessKey("access_key");

  // 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");
}
```

5.2.2 Input and Output

The input and output of ODPS GRAPH should be tables and it is not supported to self-define input and output formats.

Howerve, define job input and multiple inputs are supported:

```
GraphJob job = new GraphJob();

job.addInput(TableInfo.builder().tableName("tblname").build());   //Table as the input
job.addInput(TableInfo.builder().tableName("tblname").partSpec("pt1=a/pt2=b").build());
//Partition as the input

//only read col2 and col0 of input table. In load method of GraphLoader, the result of
record.get(0) is col2. The sequence is consistent.
job.addInput(TableInfo.builder().tableName("tblname").partSpec("pt1=a/pt2=b").build(),
new String[]{"col2", "col0"});
```

📖 **Note:**

- About the definition of job input, please refer to addInput description of GarphJob for more information. Framework reads the records of input table and transfers them to user defined Graphloader to load graph data.
- Limitation: partition filter conditions are not supported temporarily. For more application limitations, please refer to Application Limitations.

5.2.3 Read Resource

5.2.3.1 Add Resource in GRAPH Program

Besides using jar command to specify GRAPH read resource, the following methods of GraphJob can also be used:

```
void addCacheResources(String resourceNames)
void addCacheResourcesToClassPath(String resourceNames)
```

5.2.3.2 Use Resource in GRAPH Program

In graph program, use the following methods of corresponding context object 'WorkerContext' to read resource:

```
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:

- Usually the recources are read in setup method of WorkerComputer and saved in Worker Value, and then gotten in getWorkerValue method;
- Stream mentioned above is suggested, which can process the resources being read to make the memory consumption even less.
- For more application limitations, please refer to Application Limitations.

5.3 Example Programs

5.3.1 Single Source Shortest Path

Dijkstra algorithm is the classic algorithm to solve single source shortest path (Single Source Shortest Path, abbreviation: SSSP) in directed graph. The input of this algorithm contains a weighted directed graph G and a source vertex s in G. The basic principles of this algorithm include:

- Initialization: the distance from source vertex s to s itself (d[s]=0) and the distance from other vertex u to s is infinite (d[u]= ∞).
- Iteration: if there is an edge from u to v, then the shortest distance from s to v will be updated to'd[v] =min (d[v], d[u] +weight (u, v))'. Until the distances from all vertices to s do not change, the iteration is terminated.

According to the fundamentals of this algorithm, it is very suitable to implement it by OPDS Graph: each vertex maintains the shortest distance value from the source vertex. Once this value changes, send the new value added edge weights to neighbor nodes. In the following iteration, the neighbor nodes will update the current shortest distance according to their received message respectively. When the current shortest distances of all vertices do not change, the iteration will be terminated.

5.3.1.1 Source Code

```
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 combinedMessage,
        LongWritable messageToCombine) throws IOException {
      if (combinedMessage.get() > messageToCombine.get()) {
        combinedMessage.set(messageToCombine.get());
      }
    }

  }

  public static class SSSPVertexReader extends
      GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {

    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, LongWritable, LongWritable>
context)
        throws IOException {
      SSSPVertex vertex = new SSSPVertex();
      vertex.setId((LongWritable) record.get(0));
```

```
        String[] edges = record.get(1).toString().split(",");
        for (int i = 0; i < edges.length; i++) {
          String[] ss = edges[i].split(":");
          vertex.addEdge(new LongWritable(Long.parseLong(ss[0])),
              new LongWritable(Long.parseLong(ss[1])));
        }

        context.addVertexRequest(vertex);
      }

    }

    public static void main(String[] args) throws IOException {
      if (args.length < 2) {
        System.out.println("Usage: <startnode> <input> <output>");
        System.exit(-1);
      }

      GraphJob job = new GraphJob();
      job.setGraphLoaderClass(SSSPVertexReader.class);
      job.setVertexClass(SSSPVertex.class);
      job.setCombinerClass(MinLongCombiner.class);

      job.set(START_VERTEX, args[0]);
      job.addInput(TableInfo.builder().tableName(args[1]).build());
      job.addOutput(TableInfo.builder().tableName(args[2]).build());

      long startTime = System.currentTimeMillis();
      job.run();
      System.out.println("Job Finished in "
          + (System.currentTimeMillis() - startTime) / 1000.0 + " seconds");
    }
  }
```

5.3.1.2 Code Description

The source codes of SSSP include the following parts:

- Line 85: define the class SSSPVertexReader, to load graph and parse each record in table to a vertex. The first column of record is vertex ID and the second column of record is to save all edges which take this vertex as the starting point, the content is shown as: 2:2, 3:1, 4:4.
- Line 21: define SSSPVertex, in which:
- The vertex value indicates the current shortest path from this vertex to source vertex startVertexId.

- The method ‘compute()’ uses iterative formula ‘d[v]=min(d[v], d[u]+weight(u, v))’ to update vertex value;
- The method ‘cleanup()’ writes the vertex and its shortest path to source vertex into the result table;

- Line 60: when the vertex value does not change, call ‘voteToHalt ()’ to tell the framework that this vertex has entered halt status. If all vertices enter halt status, the computation is terminated.
- Line 72: define MinLongCombiner. Combine messages sent to the same vertex to optimize the performance and reduce memeory occupy.
- Line 108: main program (main function), define GraphJob, to achieve Vertex/GraphLoader/Combiner and to specify input and output tables.

5.3.2 PageRank

PageRank algorithm is a classical algorithm for computing Webpage rank. The input of this algorithm is a directed graph G and the vertex indicates webpage. If the link from webpage A to webpage B exists, then the edge from A to B also exists. The basic principle of the algorithm:

- Initialization: Vertex value indicates the rank value of PageRank (double type). At initial time, values of all vertices are ‘1/TotalNumVertices’.
- Iterative formula: PageRank (i)=0.15/TotalNumVertices+0.85*sum, in which, ‘sum’ indicates the accumulated value of PageRank (j)/out_degree (j) of all vertices (set as j) which point to the vertex i.

Seen from the basic principle of algorithm, this algorithm is very suitable for the use of ODPS Graph: each vertex (j) maintains its PageRank value. In each round of iteration, PageRank (j)/out_degree (j) will be sent to its neighbor vertex (vote for it). In next iteration, PageRank value of each vertax will be re-calculated according to iterative formula.

5.3.2.1 Source Code

```
import java.io.IOException;

import org.apache.log4j.Logger;

import com.aliyun.odps.io.WritableRecord;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.DoubleWritable;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.io.Text;
import com.aliyun.odps.io.Writable;

public class PageRank {

  private final static Logger LOG = Logger.getLogger(PageRank.class);

  public static class PageRankVertex extends
      Vertex<Text, DoubleWritable, NullWritable, DoubleWritable> {

    @Override
    public void compute(
        ComputeContext<Text, DoubleWritable, NullWritable, DoubleWritable> context,
        Iterable<DoubleWritable> messages) throws IOException {
      if (context.getSuperstep() == 0) {
        setValue(new DoubleWritable(1.0 / context.getTotalNumVertices()));
      } else if (context.getSuperstep() >= 1) {
        double sum = 0;
        for (DoubleWritable msg : messages) {
          sum += msg.get();
        }
        DoubleWritable vertexValue = new DoubleWritable(
            (0.15f / context.getTotalNumVertices()) + 0.85f * sum);
        setValue(vertexValue);
      }
      if (hasEdges()) {
        context.sendMessageToNeighbors(this, new DoubleWritable(getValue()
            .get() / getEdges().size()));
      }
    }

    @Override
    public void cleanup(
        WorkerContext<Text, DoubleWritable, NullWritable, DoubleWritable> context)
        throws IOException {
      context.write(getId(), getValue());
    }
  }
```

```
public static class PageRankVertexReader extends
    GraphLoader<Text, DoubleWritable, NullWritable, DoubleWritable> {

  @Override
  public void load(
      LongWritable recordNum,
      WritableRecord record,
      MutationContext<Text, DoubleWritable, NullWritable, DoubleWritable> context)
      throws IOException {
    PageRankVertex vertex = new PageRankVertex();
    vertex.setValue(new DoubleWritable(0));
    vertex.setId((Text) record.get(0));
    System.out.println(record.get(0));

    for (int i = 1; i < record.size(); i++) {
      Writable edge = record.get(i);
      System.out.println(edge.toString());
      if (!(edge.equals(NullWritable.get()))) {
        vertex.addEdge(new Text(edge.toString()), NullWritable.get());
      }
    }
    LOG.info("vertex 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");
  }
```

5.3.2.2 Code Description

The source codes of PageRank include the following parts:

- Line 57: define the class PageRankVertexReader to load graph and parse each record to be a vertex. The first column of record is the starting vertax and other columns are end vertices.
- Line 25: define PageRankVertex, in which:
- The vertex value indicates the current PageRank value of this vertex (webpage);
- The method compute() uses iterative formula ‘PageRank(i)=0.15/TotalNumVertices+0.85*sum’ to update the vertex value;
- The method ‘cleanup ()’ writes the vertex value and PageRank value into result table.
- Line 90: main program (main function). Define GraphJob, to specify Vertex/GraphLoader achievement, maximum iteration occurrence (default is 30) and to specify input and output tables.

5.3.3 Kmeans

Kmeans algorithm is a very basic clustering algorithm and has been widely used. The basic idea of Kmeans algorithm is: cluster by taking k vertices in space as the certer and classify the nearest vertices. Through iteration, update the clustering center value successively, to get the best clustering result.

Suppose that you want to divide a sample set into k categories and the algorithm is described as follows:

- Select the initial center of K classes properly;
- In the ith iteration, calculate the distances from a random sample to k centers and classify the sample into the categories where the center of shortest

distance lies.

- Use mean value or other methods to update center value of this class.
- For all clustering centers (quantity is k), if they are updated through the iteration method mentioned in last two steps and their values keep unchanged or are less than a threshold, then the iteration is terminated. Otherwise, continue the iteration.

5.3.3.1 Source Code

```
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.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.Tuple;
import com.aliyun.odps.io.Writable;

public class Kmeans {
  private final static Logger LOG = Logger.getLogger(Kmeans.class);

  public static class KmeansVertex extends
      Vertex<Text, Tuple, NullWritable, NullWritable> {

    @Override
    public void compute(
        ComputeContext<Text, Tuple, NullWritable, NullWritable> context,
        Iterable<NullWritable> messages) throws IOException {
      context.aggregate(getValue());
    }
```

```
}

public static class KmeansVertexReader extends
    GraphLoader<Text, Tuple, NullWritable, NullWritable> {
  @Override
  public void load(LongWritable recordNum, WritableRecord record,
      MutationContext<Text, Tuple, NullWritable, NullWritable> context)
      throws IOException {
    KmeansVertex vertex = new KmeansVertex();
    vertex.setId(new Text(String.valueOf(recordNum.get())));
    vertex.setValue(new Tuple(record.getAll()));
    context.addVertexRequest(vertex);
  }

}

public static class KmeansAggrValue implements Writable {

  Tuple centers = new Tuple();
  Tuple sums = new Tuple();
  Tuple counts = new Tuple();

  @Override
  public void write(DataOutput out) throws IOException {
    centers.write(out);
    sums.write(out);
    counts.write(out);
  }

  @Override
  public void readFields(DataInput in) throws IOException {
    centers = new Tuple();
    centers.readFields(in);
    sums = new Tuple();
    sums.readFields(in);
    counts = new Tuple();
    counts.readFields(in);
  }

  @Override
  public String toString() {
    return "centers " + centers.toString() + ", sums " + sums.toString()
        + ", counts " + counts.toString();
  }

}

public static class KmeansAggregator extends Aggregator<KmeansAggrValue> {

  @SuppressWarnings("rawtypes")
  @Override
  public KmeansAggrValue createInitialValue(WorkerContext context)
      throws IOException {
    KmeansAggrValue aggrVal = null;
    if (context.getSuperstep() == 0) {
      aggrVal = new KmeansAggrValue();
      aggrVal.centers = new Tuple();
      aggrVal.sums = new Tuple();
      aggrVal.counts = new Tuple();

      byte[] centers = context.readCacheFile("centers");
      String lines[] = new String(centers).split("\n");

      for (int i = 0; i < lines.length; i++) {
        String[] ss = lines[i].split(",");
        Tuple center = new Tuple();
        Tuple sum = new Tuple();
        for (int j = 0; j < ss.length; ++j) {
          center.append(new DoubleWritable(Double.valueOf(ss[j].trim())));
          sum.append(new DoubleWritable(0.0));
        }
        LongWritable count = new LongWritable(0);
        aggrVal.sums.append(sum);
        aggrVal.counts.append(count);
        aggrVal.centers.append(center);
      }
    } else {
      aggrVal = (KmeansAggrValue) context.getLastAggregatedValue(0);
    }

    return aggrVal;
  }

  @Override
  public void aggregate(KmeansAggrValue value, Object item) {
    int min = 0;
    double mindist = Double.MAX_VALUE;
```

```
        Tuple point = (Tuple) item;

        for (int i = 0; i < value.centers.size(); i++) {
            Tuple center = (Tuple) value.centers.get(i);
            // use Euclidean Distance, no need to calculate sqrt
            double dist = 0.0d;
            for (int j = 0; j < center.size(); j++) {
                double v = ((DoubleWritable) point.get(j)).get()
                        - ((DoubleWritable) center.get(j)).get();
                dist += v * v;
            }
            if (dist < mindist) {
                mindist = dist;
                min = i;
            }
        }

        // update sum and count
        Tuple sum = (Tuple) value.sums.get(min);
        for (int i = 0; i < point.size(); i++) {
            DoubleWritable s = (DoubleWritable) sum.get(i);
            s.set(s.get() + ((DoubleWritable) point.get(i)).get());
        }
        LongWritable count = (LongWritable) value.counts.get(min);
        count.set(count.get() + 1);
    }

    @Override
    public void merge(KmeansAggrValue value, KmeansAggrValue partial) {
        for (int i = 0; i < value.sums.size(); i++) {
            Tuple sum = (Tuple) value.sums.get(i);
            Tuple that = (Tuple) partial.sums.get(i);

            for (int j = 0; j < sum.size(); j++) {
                DoubleWritable s = (DoubleWritable) sum.get(j);
                s.set(s.get() + ((DoubleWritable) that.get(j)).get());
            }
        }

        for (int i = 0; i < value.counts.size(); i++) {
            LongWritable count = (LongWritable) value.counts.get(i);
            count.set(count.get() + ((LongWritable) partial.counts.get(i)).get());
        }
    }
```

```
@SuppressWarnings("rawtypes")
@Override
public boolean terminate(WorkerContext context, KmeansAggrValue value)
    throws IOException {

  // compute new centers
  Tuple newCenters = new Tuple(value.sums.size());
  for (int i = 0; i < value.sums.size(); i++) {
    Tuple sum = (Tuple) value.sums.get(i);
    Tuple newCenter = new Tuple(sum.size());
    LongWritable c = (LongWritable) value.counts.get(i);
    for (int j = 0; j < sum.size(); j++) {

      DoubleWritable s = (DoubleWritable) sum.get(j);
      double val = s.get() / c.get();
      newCenter.set(j, new DoubleWritable(val));

      // reset sum for next iteration
      s.set(0.0d);
    }
    // reset count for next iteration
    c.set(0);
    newCenters.set(i, newCenter);
  }

  // update centers
  Tuple oldCenters = value.centers;
  value.centers = newCenters;

  LOG.info("old centers: " + oldCenters + ", new centers: " + newCenters);

  // compare new/old centers
  boolean converged = true;
  for (int i = 0; i < value.centers.size() && converged; i++) {
    Tuple oldCenter = (Tuple) oldCenters.get(i);
    Tuple newCenter = (Tuple) newCenters.get(i);
    double sum = 0.0d;
    for (int j = 0; j < newCenter.size(); j++) {
      double v = ((DoubleWritable) newCenter.get(j)).get()
          - ((DoubleWritable) oldCenter.get(j)).get();
      sum += v * v;
    }
    double dist = Math.sqrt(sum);
```

```
                LOG.info("old center: " + oldCenter + ", new center: " + newCenter
                    + ", dist: " + dist);
                // converge threshold for each center: 0.05
                converged = dist < 0.05d;
            }

            if (converged || context.getSuperstep() == context.getMaxIteration() - 1) {
                // converged or reach max iteration, output centers
                for (int i = 0; i < value.centers.size(); i++) {
                    context.write(((Tuple) value.centers.get(i)).toArray());
                }
                // true means to terminate iteration
                return true;
            }

            // false means to continue iteration
            return false;
        }
    }

    private static void printUsage() {
        System.out.println("Usage: <in> <out> [Max iterations (default 30)]");
        System.exit(-1);
    }

    public static void main(String[] args) throws IOException {
        if (args.length < 2)
            printUsage();

        GraphJob job = new GraphJob();

        job.setGraphLoaderClass(KmeansVertexReader.class);
        job.setRuntimePartitioning(false);
        job.setVertexClass(KmeansVertex.class);
        job.setAggregatorClass(KmeansAggregator.class);
        job.addInput(TableInfo.builder().tableName(args[0]).build());
        job.addOutput(TableInfo.builder().tableName(args[1]).build());

        // default max iteration is 30
        job.setMaxIteration(30);
        if (args.length >= 3)
            job.setMaxIteration(Integer.parseInt(args[2]));

        long start = System.currentTimeMillis();
        job.run();
        System.out.println("Job Finished in "
            + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
    }
}
```

5.3.3.2 Code Description

The source codes of Kmeans include the following parts:

- Line 40: define the class KmeansVertexReader, to load graph and parse each record in table to be a vertex. The vertex ID is of no great importance. Here take the incoming 'recordNum' as the vertex ID and the vertex value is Tuple composed of all columns.
- Line 32: define the class KmeansVertex. The method compute() is very simple, only to call aggregate method of context objects and income the value of current vertex (Tuple type, indicated by vector).
- Line 85: define the class KmeansAggregator, which encapsulates the main logic of Kmeans algorithm. In which:
- createInitialValue creates initial value for each round of iteration. If it is the first round iteration (superstep=0), this value is the initial center vertex, otherwise the value is the new center vertex on the end of last iteration.
- The method 'aggregate' calculates the distance from each vertex to each center as the shortest distance class and update sum and count of this class.
- The method 'merge' is to combine sum and count collected from each worker.
- The method 'terminate' is to calculate new center vertex according to sum and count of each class. If the distance from new center vertex to old center vertex is less than a certain threshold or the iteration occerence reaches maximum iteration occerence which has been set, then the iteration is terminated (return false). The final center vertex will be written into result table.
- Line 238: main program (main function), to define GraphJob and specify the Vertex/GraphLoader/Aggregator achievements, maximum iteration occerence (default is 30) and input/output table.
- Line 245: job.setRuntimePartitioning (false), for Kmeans algorithm, loading graph does not need vertices distribution. Set RuntimePartitioning to be false, to improve the performance of loading diagram.

5.3.4 BipartiteMatching

Bipartite graph refers to that all the vertices of graph can be divided into two sets and the two vertices corresponded by each edge belong to these two sets. For a bipartite G (M is its subgraph), if any two edges of M are not attached to the same vertex, and then M is called a matching. Bipartite graph is usually used for information matching in the scene that has clear supplay and demand relationship (such as dating sites, etc.).

Algorithm description is shown as follows:

- Beginning from the first vertex on the left, choose unmatched vertices and start searching to search augmenting path.
- If an unmatched vertex is found, it indicates that the searching is successful.
- Update the path information, matched by edges+1. Stop searching.
- If the augmenting path has not been found, then searching is no longer from this vertex.

5.3.4.1 Source Code

```
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<Text> 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> [maxIteration]");
  }

  public static void main(String[] args) throws IOException {
    if (args.length < 2) {
      printUsage();
    }

    GraphJob job = new GraphJob();

    job.setGraphLoaderClass(BipartiteMatchingVertexReader.class);
    job.setVertexClass(BipartiteMatchingVertex.class);

    job.addInput(TableInfo.builder().tableName(args[0]).build());
    job.addOutput(TableInfo.builder().tableName(args[1]).build());
    int maxIteration = 30;
    if (args.length > 2) {
      maxIteration = Integer.parseInt(args[2]);
    }
    job.setMaxIteration(maxIteration);

    job.run();
  }

}
```

5.3.5 Strongly Connected Components

In the mathematical theory of directed graphs, a graph is said to be strongly connected if every vertex is reachable from other vertex. The strongly connected components of an arbitrary directed graph form a partition into subgraphs that are themselves strongly connected. The example of this algorithm is based on parallel

Coloring algorithm.

Each vertex includes two parts:

- colorID: in the forward-traversal process, the colors of vertex v are stored. After the calculation is terminated, the vertices with the same colorID belong to a strongly connected component.
- transposeNeighbors: store the Neighbor ID of vertex v in the transposed graph of input graph.

The algorithm includes the following steps:

- Transpose Graph Formation: Requires two supersteps. In the first superstep, each vertex sends a message with its ID to all its outgoing neighbors, which in the second superstep are stored in transposeNeighbors.
- 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.
- 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.
- Backward-Traversal: We break the phase into Start and Rest again. 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 vertices are inactive.

5.3.5.1 Source 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.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
 * strongly connected.
 *
 * Algorithms with four phases as follows.
 * 1. Transpose Graph Formation: Requires two supersteps. In the first
 * superstep, each vertex sends a message with its ID to all its outgoing
 * neighbors, which in the second superstep are stored in transposeNeighbors.
 *
 * 2. Trimming: Takes one superstep. Every vertex with only in-coming or only outgoing
edges (or neither) sets its colorID to its own ID and becomes inactive. Messages
subsequently sent to the vertex are ignored.
 *
 * 3. Forward-Traversal: There are two sub phases: Start and Rest. In the
 * Start phase, each vertex sets its colorID to its own ID and propagates
 * its ID to its outgoing neighbors. In the Rest phase, vertices update
 * their own colorIDs with the minimum colorID they have seen, and propagate
 * their colorIDs, if updated, until the colorIDs converge.
 * Set the phase to Backward-Traversal when the colorIDs converge.
 *
 * 4. Backward-Traversal: We again break the phase into Start and Rest.
 * In Start, every vertex whose ID equals its colorID propagates its ID to
 * the vertices in transposeNeighbors and sets itself inactive. Messages
 * subsequently sent to the vertex are ignored. In each of the Rest phase supersteps,
 * each vertex receiving a message that matches its colorID: (1) propagates
 * its colorID in the transpose graph; (2) sets itself inactive. Messages
 * subsequently sent to the vertex are ignored. Set the phase back to Trimming
 * if not all vertex are inactive.
 *
 * http://ilpubs.stanford.edu:8090/1077/3/p535-salihoglu.pdf
 */
public class StronglyConnectedComponents {

  public final static int STAGE_TRANSPOSE_1 = 0;
  public final static int STAGE_TRANSPOSE_2 = 1;
  public final static int STAGE_TRIMMING = 2;
  public final static int STAGE_FW_START = 3;
  public final static int STAGE_FW_REST = 4;
  public final static int STAGE_BW_START = 5;
  public final static int STAGE_BW_REST = 6;

  /**
   * The value is composed of component id, incoming neighbors,
   * active status and updated status.
   */
  public static class MyValue implements Writable {

    LongWritable sccID;// strongly connected component id
    Tuple inNeighbors; // transpose neighbors

    BooleanWritable active; // vertex is active or not
    BooleanWritable updated; // sccID is updated or not

    public MyValue() {
      this.sccID = new LongWritable(Long.MAX_VALUE);
      this.inNeighbors = new Tuple();
      this.active = new BooleanWritable(true);
      this.updated = new BooleanWritable(false);
    }

    public void setSccID(LongWritable sccID) {
      this.sccID = sccID;
    }

    public LongWritable getSccID() {
      return this.sccID;
    }

    public void setInNeighbors(Tuple inNeighbors) {
      this.inNeighbors = inNeighbors;
    }

    public Tuple getInNeighbors() {
      return this.inNeighbors;
    }

    public void addInNeighbor(LongWritable neighbor) {
      this.inNeighbors.append(new LongWritable(neighbor.get()));
    }

    public boolean isActive() {
      return this.active.get();
    }

    public void setActive(boolean status) {
      this.active.set(status);
    }

    public boolean isUpdated() {
      return this.updated.get();
    }

    public void setUpdated(boolean update) {
      this.updated.set(update);
    }

    @Override
    public void write(DataOutput out) throws IOException {
      this.sccID.write(out);
      this.inNeighbors.write(out);
      this.active.write(out);
      this.updated.write(out);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
      this.sccID.readFields(in);
      this.inNeighbors.readFields(in);
      this.active.readFields(in);
      this.updated.readFields(in);
    }

    @Override
    public String toString() {
      StringBuilder sb = new StringBuilder();
      sb.append("sccID: " + sccID.get());
      sb.append(" 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, LongWritable> 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");

    }

  }
```

5.3.6 Connected Components

If the path exists between two vertices, we consider these two vertices to be connected. If arbitrary two vertices in undirected graph G are connected, then G is called connected graph, otherwise it is called unconnected graph. The subs connected graph with large number of vertices is called connected component. This algorithm is to compute the connected component membership of each vertex, output each vertex which's value containing the smallest vertex id in the connected component containing that vertex finally and propagate the smallest vertex id along the edges to all vertices of a connected component.

5.3.6.1 Source Code

```
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, LongWritable>
context)
          throws IOException {
        context.write(getId(), getValue());
      }
    }


    /**
     * Input Table Description:
     * +-----------------+----------------------------------------------------+
     * | Field | Type       | Comment                                                      |
     * +-----------------+----------------------------------------------------+
     * | v          | bigint    | vertex id                                                    |
     * | es        | string    | comma separated target vertex id of outgoing edges |
     * +-----------------+----------------------------------------------------+
     *
     * Example:
```

```
   * For graph:
   *           1 ----- 2
   *           |           |
   *           3 ----- 4
   * Input table:
   * +-----------+
   * | 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");
    }
}
```

5.3.7 Topological Sorting

For the directed edges (u, v), define the sequence of all vertices satisfying the condition (u<v) to be topological sequence. Topological sorting is an algorithm to seek the topological sequence for a directed graph.

Algorithm:

- Find a vertex which has no incoming edge from the graph and output it.
- Delete this vertex and its all outgoing edges from the graph.
- Repeat the steps mentioned above, until all vertices have been output.

5.3.7.1 Source Code

```
import java.io.IOException;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.graph.Aggregator;
import com.aliyun.odps.graph.Combiner;
import com.aliyun.odps.graph.ComputeContext;
import com.aliyun.odps.graph.GraphJob;
import com.aliyun.odps.graph.GraphLoader;
import com.aliyun.odps.graph.MutationContext;
import com.aliyun.odps.graph.Vertex;
import com.aliyun.odps.graph.WorkerContext;
import com.aliyun.odps.io.LongWritable;
import com.aliyun.odps.io.NullWritable;
import com.aliyun.odps.io.BooleanWritable;
import com.aliyun.odps.io.WritableRecord;

public class TopologySort {

  private final static Log LOG = LogFactory.getLog(TopologySort.class);

  public static class TopologySortVertex extends
      Vertex<LongWritable, LongWritable, NullWritable, LongWritable> {

    @Override
    public void compute(
        ComputeContext<LongWritable, LongWritable, NullWritable, LongWritable>
context,
        Iterable<LongWritable> messages) throws IOException {
      // in superstep 0, each vertex sends message whose value is 1 to its
      // neighbors
      if (context.getSuperstep() == 0) {
        if (hasEdges()) {
          context.sendMessageToNeighbors(this, new LongWritable(1L));
        }
      } else if (context.getSuperstep() >= 1) {
        // compute each vertex's indegree
        long indegree = getValue().get();
        for (LongWritable msg : messages) {
          indegree += msg.get();
        }
        setValue(new LongWritable(indegree));
        if (indegree == 0) {
          voteToHalt();
          if (hasEdges()) {
            context.sendMessageToNeighbors(this, new LongWritable(-1L));
          }
          context.write(new LongWritable(context.getSuperstep()), getId());
          LOG.info("vertex: " + getId());
        }
        context.aggregate(new LongWritable(indegree));
      }
    }
  }

  public static class TopologySortVertexReader extends
      GraphLoader<LongWritable, LongWritable, NullWritable, LongWritable> {
```

```
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, NullWritable, LongWritable>
context)
        throws IOException {
      TopologySortVertex vertex = new TopologySortVertex();
      vertex.setId((LongWritable) record.get(0));
      vertex.setValue(new LongWritable(0));

      String[] edges = record.get(1).toString().split(",");
      for (int i = 0; i < edges.length; i++) {
        long edge = Long.parseLong(edges[i]);
        if (edge >= 0) {
          vertex.addEdge(new LongWritable(Long.parseLong(edges[i])),
              NullWritable.get());
        }
      }
      LOG.info(record.toString());
      context.addVertexRequest(vertex);
    }

  }

  public static class LongSumCombiner extends
      Combiner<LongWritable, LongWritable> {

    @Override
    public void combine(LongWritable vertexId, LongWritable combinedMessage,
        LongWritable messageToCombine) throws IOException {
      combinedMessage.set(combinedMessage.get() + messageToCombine.get());
    }

  }

  public static class TopologySortAggregator extends
      Aggregator<BooleanWritable> {

    @SuppressWarnings("rawtypes")
    @Override
    public BooleanWritable createInitialValue(WorkerContext context)
        throws IOException {
```

```
        return new BooleanWritable(true);
      }

      @Override
      public void aggregate(BooleanWritable value, Object item)
          throws IOException {
        boolean hasCycle = value.get();
        boolean inDegreeNotZero = ((LongWritable) item).get() == 0 ? false : true;
        value.set(hasCycle && inDegreeNotZero);
      }

      @Override
      public void merge(BooleanWritable value, BooleanWritable partial)
          throws IOException {
        value.set(value.get() && partial.get());
      }

      @SuppressWarnings("rawtypes")
      @Override
      public boolean terminate(WorkerContext context, BooleanWritable value)
          throws IOException {
        if (context.getSuperstep() == 0) {
          // since the initial aggregator value is true, and in superstep we don't
          // do aggregate
          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 format of input table is:
      // 0 1，2
      // 1 3
      // 2 3
      // 3 -1
      // the first column is vertexid and the second column is destination vertexid of the
 outgoing edge. If the value is -1, it indicates that this vertex has no outgoing edge.
```

```
    // the format of output table is:
    // 0 0
    // 1 1
    // 1 2
    // 2 3
    // the first column is the value of 'supstep', impliciting topological sequence; the second
column is vertexid.
    // TopologySortAggregator is used to judge whether there is loop in this graph.
    // If there is loop in input graph,then the iteration is terminated when the penetrations of
vertices in active status are not 0.
    // User can judge whether a directed graph has loop through the record number of input
table and output table.
    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");
  }
}
```

5.3.8 Linear Regression

In statistics, linear regression is a statistical analysis method used to determine the interdependence relationship between two variables or among more than two variables. Differently from classification algorithm to discrete prediction, regression algorithm can predict successive value type. Linear regression algorithm defines the lost function as the minimum square error sum of sample set to solve the weight vector through minimizing the lost function. The commonly used method is the gradient descent method:

- Initialize the weight vector and give the descent rate and the iteration occerence (or the iteration convergence condition);
- For each piece of sample, calculate minimum square error.
- Get the sum of minimum square error and update the weights according to the descent rate.
- Repeat the iterations until convergence.

5.3.8.1 Source 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,......
 *
 * @author shiwan.ch
 * @update jiasheng.tjs running parameters are like: tjs_lr_in tjs_lr_out 1500 2
 *            0.07
 */

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_Num")) ==
gradient.count
            .get()) {
```

```
            context.write(gradient.currentTheta.toArray());
            return true;
          }
          gradient.lastTheta = tmpCurrentTheta;
          gradient.currentTheta = tmpCurrentTheta;
          gradient.count.set(gradient.count.get() + 1);
          int n = (int) Long.parseLong(context.getConfiguration().get("Dimension"));
          /*
           * update 4: Important!!! Remember this step. Graph won't reset the
           * initial value for global variables at the beginning of each iteration
           */
          for (int i = 0; i < n; i++) {
            gradient.tmpGradient.set(i, new DoubleWritable(0));
          }
          return false;
        }

      }

      public static void main(String[] args) throws IOException {
        GraphJob job = new GraphJob();

        job.setGraphLoaderClass(LinearRegressionVertexReader.class);
        job.setRuntimePartitioning(false);
        job.setNumWorkers(3);
        job.setVertexClass(LinearRegressionVertex.class);
        job.setAggregatorClass(LinearRegressionAggregator.class);
        job.addInput(TableInfo.builder().tableName(args[0]).build());
        job.addOutput(TableInfo.builder().tableName(args[1]).build());
        job.setMaxIteration(Integer.parseInt(args[2])); // Numbers of Iteration
        job.setInt("Max_Iter_Num", Integer.parseInt(args[2]));
        job.setInt("Dimension", Integer.parseInt(args[3])); // Dimension
        job.setFloat("Alpha", Float.parseFloat(args[4])); // Learning rate

        long start = System.currentTimeMillis();
        job.run();
        System.out.println("Job Finished in "
            + (System.currentTimeMillis() - start) / 1000.0 + " seconds");
      }

    }
```

5.3.9 Triangle Count

Compute the number of triangles passing through each vertex.

The algorithm can be computed in three supersteps:

- Each vertex sends a message with its ID to all its outgoing neighbors.
- The incoming neighbors and outgoing neighbors are stored and sent to outgoing neighbors.
- For each edge, compute the intersection quantity of its destination vertex and sum them, then output the result into table.

Get the sum of output result in the table and divide by three. Then get the triangle number.

5.3.9.1 Source Code

```
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 divides 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");
    }
}
```

5.3.10 GraphLoader

5.3.10.1 Source 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;


/**
```

```
 * This example is used to demonstrate how to write graphjob program to load data for
different data types. It mainly demonstrates the graph construction by associating
GraphLoader with VertexResolver.
 *
 * The job input of ODPS Graph is table of ODPS. Suppose that the job input has two
tables, one is to store vertex information and the other is to store edge information.
 * The table format to save vertex information is shown as follows:
 * +------------------------+
 * | VertexID | VertexValue |
 * +------------------------+
 * |           id0|                    9|
 * +------------------------+
 * |           id1|                    7|
 * +------------------------+
 * |           id2|                    8|
 * +------------------------+
 *
 * the table format to save edge information is shown as follows:
 * +-----------------------------------+
 * | VertexID | DestVertexID| EdgeValue|
 * +-----------------------------------+
 * |           id0|               id1|              1|
 * +-----------------------------------+
 * |           id0|               id2|              2|
 * +-----------------------------------+
 * |           id2|               id1|              3|
 * +-----------------------------------+
 *
 * With data of two tables, it indicates that id0 has two outgoing edges, pointing to id1 and
id2 seperately. id2 has one outgoing edge, pointing to id1; id1 has no outgoing edge.
 *
 * For this type data, in GraphLoader::load(LongWritable, Record, MutationContext), you
can use MutationContext#addVertexRequest(Vertex) to request adding vertices in graph.
Use ‘link MutationContext#addEdgeRequest(WritableComparable, Edge)’ to request
adding edges in graph. Then in ‘link VertexResolver#resolve(WritableComparable, Vertex,
VertexChanges, boolean)’, combine the added vertices and edges by ‘load’ method into a
Vertex object and add it in the last graph which is involed in computing as the return value.
 *
 **/
public class VertexInputFormat {

  private final static String EDGE_TABLE = "edge.table";

  /**
```

```
   * Phrase record into Vertex and Edge. Each Record indicates one Vertex or one Edge
according to its source.
   * <p>
   * Similarly to com.aliyun.odps.mapreduce.Mapper#map, the input record can generate
key value pairs. Here the key isVertex ID and the value is Vertex or Edge, which are
written through Context. These key value pairs will be gathered in LoadingVertexResolver
according to Vertex ID.
   *
   * Note: the added vertices or edges are just requested according to rocord contents,
not the vertices or edges which are involved in computing at last. Only the vertices and
edges added in following VertexResolver are invoived in computing.
   **/
  public static class VertexInputLoader extends
      GraphLoader<LongWritable, LongWritable, LongWritable, LongWritable> {

    private boolean isEdgeData;

    /**
     * Configure VertexInputLoader
     *
     * @param conf
     * configuration parameter of job， configured by GraphJob in main class or set in
console.
     * @param workerId
     * ID of current running worker, starting from 0, used to construct unique vertex id.
     * @param inputTableInfo
     * the information of input table loaded by current worker, can be used to confirm
which data type the current input belongs to, that is the format of record.
     **/
    @Override
    public void setup(Configuration conf, int workerId, TableInfo inputTableInfo) {
      isEdgeData = conf.get(EDGE_TABLE).equals(inputTableInfo.getTableName());
    }

    /**
     * Parse the contents in the record to corresponding edges and request to be added
in the graph.
     *
     * @param recordNum
     *Record the sequence number, starting from 1. Count the sequence number for each
worker separately.
     * @param record
     *Reocrds in input table, which inclues three columns, indicating initial vertex, end
vertex and edge weights.
```

```
     * @param context
     *it refers to the context, to request adding parsed edges into the graph.
     **/
    @Override
    public void load(
        LongWritable recordNum,
        WritableRecord record,
        MutationContext<LongWritable, LongWritable, LongWritable, LongWritable>
context)
        throws IOException {
      if (isEdgeData) {
        /**
         * the data comes from the table which saves edge information.
         *
         * 1 the first column indicates the initial vertex ID.
         **/
        LongWritable sourceVertexID = (LongWritable) record.get(0);

        /**
         * 2 the second column indicates the end vertex ID.
         **/
        LongWritable destinationVertexID = (LongWritable) record.get(1);

        /**
         * 3 the third column indicates the edge weights.
         **/
        LongWritable edgeValue = (LongWritable) record.get(2);

        /**
         * 4 Create the edge, composed of end vertex ID and edge weights.
         **/
        Edge<LongWritable, LongWritable> edge = new Edge<LongWritable,
LongWritable>(
            destinationVertexID, edgeValue);

        /**
         * 5 Send request to add edge for initial vertex.
         **/
        context.addEdgeRequest(sourceVertexID, edge);

        /**
         * 6 If each Record represents a double edge, repeat 4 and 5.
Edge<LongWritable, LongWritable> edge2 = new
         * Edge<LongWritable, LongWritable>( sourceVertexID, edgeValue);
```

```
            * context.addEdgeRequest(destinationVertexID, edge2);
            **/
        } else {
           /**
            * The data comes from the table which saves vertex information.
            *
            * 1 The first column indicates the vertex ID
            **/
           LongWritable vertexID = (LongWritable) record.get(0);

           /**
            * 2 The second column indicates the vertex values.
            **/
           LongWritable vertexValue = (LongWritable) record.get(1);

           /**
            * 3 Create the vertex, composed of the vertex ID and the vertex value.
            **/
           MyVertex vertex = new MyVertex();

           /**
            * 4 Initialize the vertex.
            **/
           vertex.setId(vertexID);
           vertex.setValue(vertexValue);

           /**
            * 5 Send request to add vertex.
            **/
           context.addVertexRequest(vertex);
        }

      }

   }

   /**
     * Summarize the key value pairs generated by GraphLoader::load(LongWritable,
Record, MutationContext), which is similar to reduce in
com.aliyun.odps.mapreduce.Reducer. For a unique Vertex ID, all behaviors as
adding/deleting vertices /edges will be put into VertexChanges.
     *
     * Note: here is not only called after adding conflicted vertices and edges in load
method. (Conflict means adding multiple same vertex objects, adding repeated edges and
so on.) All generated IDs will be called here in load method.
   **/
  public static class LoadingResolver extends
       VertexResolver<LongWritable, LongWritable, LongWritable, LongWritable> {

     /**
      * Process a request to add/delete vertex/edge about one ID.
      *
      * VertexChanges has four interfaces, corresponding to four interfaces of
MutationContext separately:
      * VertexChanges::getAddedVertexList() corresponds to
MutationContext::addVertexRequest(Vertex);
      * In load method, the vertex objects with the same ID to be added by sending
requests will be summarized in return list.
      * VertexChanges::getAddedEdgeList() corresponds to
MutationContext::addEdgeRequest(WritableComparable, Edge). The edge objects with
the same initial vertex ID to be added by sending requests will be summarized in return
list.
      * VertexChanges::getRemovedVertexCount() corresponds to
MutationContext::removeVertexRequest(WritableComparable). The vertices with the
same ID are requested to be removed. Summarize the removed vertex count as the return
value.
      * VertexChanges#getRemovedEdgeList() corresponds to
MutationContext#removeEdgeRequest(WritableComparable, WritableComparable). The
edge objects with the same initial vertex ID which are requested to be removed will be
summarized in return list.
      *
      * User declares whether this ID is involved in computing by return value and through
processing the changes of this ID. If the return vertex is not null, then this ID will be
involved in following computing. If the return value is null, then this ID will not be involved
in computing.
      *
      * @param vertexId
      * Vertex ID or initial vertex ID of edge, which is requested to be added
      * @param vertex
      * Vertex object that has existed. In data load stage, it is always NULL.
      * @param vertexChanges
      * the set of vertices /edges to be added/removed, corresponding to this ID
      * @param hasMessages
      * Whether this ID has input message. In data load stage, it is always 'false'.
      **/
     @Override
     public Vertex<LongWritable, LongWritable, LongWritable, LongWritable> resolve(
          LongWritable vertexId,
        Vertex<LongWritable, LongWritable, LongWritable, LongWritable> vertex,
        VertexChanges<LongWritable, LongWritable, LongWritable, LongWritable>
vertexChanges,
        boolean hasMessages) throws IOException {
      /**
       * 1 Get Vertex objects, as the vertices which are participated in computing.
       **/
      MyVertex computeVertex = null;
      if (vertexChanges.getAddedVertexList() == null
          || vertexChanges.getAddedVertexList().isEmpty()) {
        computeVertex = new MyVertex();
        computeVertex.setId(vertexId);
      } else {
        /**
         * Here suppose that every record in the table which saves vertex information
indicates a unique vertex.
         **/
        computeVertex = (MyVertex) vertexChanges.getAddedVertexList().get(0);
      }

      /**
       * 2 Send the request to the edge added by this vertex and add it into Vertex object.
If the data is possible to be repeated, you can determine whether to do duplicate removal
according to the algorithm.
       **/
      if (vertexChanges.getAddedEdgeList() != null) {
        for (Edge<LongWritable, LongWritable> edge : vertexChanges
            .getAddedEdgeList()) {
          computeVertex.addEdge(edge.getDestVertexId(), edge.getValue());
        }
      }

      /**
       * 3 Return Vertex objects and add them into final graph to participate in computing.
       **/
      return computeVertex;
    }

  }

  /**
   * Determine the behavior of vertices which are participated in computing.
   *
   **/
  public static class MyVertex extends
      Vertex<LongWritable, LongWritable, LongWritable, LongWritable> {

    /**
     * Write the edges of vertex into result table according to the input table format. The
formats and data of input table and output table are the same.
     *
     * @param context
     * the context in running process
     * @param messages
     * Input message
     **/
    @Override
    public void compute(
        ComputeContext<LongWritable, LongWritable, LongWritable, LongWritable>
context,
        Iterable<LongWritable> messages) throws IOException {
      /**
       * Write vertex ID and value into the result table which saves vertices.
       **/
      context.write("vertex", getId(), getValue());

      /**
       * Write the edges of vertex into the result table which saves edges.
       **/
      if (hasEdges()) {
        for (Edge<LongWritable, LongWritable> edge : getEdges()) {
          context.write("edge", getId(), edge.getDestVertexId(),
              edge.getValue());
        }
      }

      /**
       * It only needs one round 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 用于对 Graph 作业进行配置
     */
    GraphJob job = new GraphJob();

    /**
     * 1 Specify input graph data and specify the table which saves edge information.
     */
    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 method to load data. Parse Record to Edge, which is similar to
map.The generated key is vertex ID and the key value is edge.
     */
    job.setGraphLoaderClass(VertexInputLoader.class);

    /**
     * 3 Specify data load stage and generate the vertices to be participated in computing.
Here is similar to reduce, to combine the generated edges by mapping into a vertex.
     */
    job.setLoadingVertexResolverClass(LoadingResolver.class);

    /**
     * 4 Specify the behavior of vertices which are involved in computing.Run the method
vertex.compute in each iteration.
     */
    job.setVertexClass(MyVertex.class);

    /**
     * 5 Specify the output table of graph job and write the generated result by computing
into result table.
     */
    job.addOutput(TableInfo.builder().tableName(args[2]).label("vertex").build());
    job.addOutput(TableInfo.builder().tableName(args[3]).label("edge").build());
```

```
        /**
         * 6 Submit job to run.
         */
        job.run();
    }

}
```

5.4 SDK Introduction

Main SDK	Description
GraphJob	GraphJob is inherited from JobConf, used to define, submit and manage an ODPS Graph job.
Vertex	Vertex refers to the vertex of a graph, including the following properties: id, value, halted and edges. It provides Vertex through setVertexClass API of GraphJob.
Edge	Edge refers to the edge of a graph, including the following properties: destVertexId, value. Graph data structure uses the adjacency table and the outgoing edges of a vertex are saved in edges of this vertex.
GraphLoader	GraphLoader is used to load a graph. It provides GraphLoader through setGraphLoaderClass API of GraphJob.
VertexResolver	VertexResolver is used to define the conflict processing logic while the graph topology changes. The conflict processing logic while the graph topology changes in graph loading and iteration computing courses is provided through setLoadingVertexResolverClass and setComputingVertexResolverClass of GraphJob.
Partitioner	Partitioner is used to devide the graph and make the computing run in sclices. Partitioner is provided through setPartitionerClass of GraphJob and the default is HashPartitioner. That is to get hash values of vertex ID and then obtain modulo of Worker number.
WorkerComputer	WorkerComputer is allowed to execute user defined logic while the worker starts and quits. WorkerComputer is provided through

Main SDK	Description
	setWorkerComputerClass of GraphJob.
Aggregator	SetAggregatorClass (Class ...) of Aggregator, to define one or more Aggregators.
Combiner	setCombinerClass of combiner, to set Combiner
Counters	The counter can be gotten through WorkerContext and starts to count in job running logic. The framework will summarize the result automatically.
WorkerContext	The context object, it encapsulates the functions provided by framework, such as modifying graph topology, sending messages, writing results, reading resources, etc.

5.5 Application Limitations

- The number of resources quoted by a single job can not exceed 256. Table and archive are considered as one unit.
- The size of total bytes quoted by a single job can not exceed 512 M.
- The number of input channels of a single job cannot exceed 1024 (the number of input tables cannot exceed 64.); the number of output channels of a single job cannot exceed 256.
- The label specified in multiple outputs cannot be null or a empty string. Its length cannot exceed 256 and it can only contain A-Z, a-z, 0-9, _, #, - and so on.
- The number of counters defined in a single job cannot exceed 64. The group name and counter name cannot contain '#' and the length sum of them cannot exceed 100.
- The worker number of a single job is calculated by framework and the maximum is 1000, otherwise the expection will be thrown.
- The default CPU size occupied by a single job is 200 and the range is [50, 800].
- The default memory size occupied by a single job is 4096 and the range is [256M, 12G].
- The occurrence that a single worker reads a resource repeatedly cannot be more than 64.

- The default plit size is 64M, which can be set by users and the range is: 0 < split_size <= (9223372036854775807 >> 20).
- In the cluster running process, GraphLoader/Vertex/Aggregator in ODPS Graph program are limited by Java sandbox. (the main program of Graph job has no this limitation.)

6 Tools

- ☐ ODPS Client
- ☐ Eclipse Plug-in

6.1 ODPS Client

6.1.1 Summary

6.1.1.1 Declaration

- This section is a part of ODPS client user manual, to explain how to use the basic functions of ODPS services by means of command line tool.
- Please do not rely on the output format of client to do any analysis work. The forward compatibility is not promised by the output format of client. The command formats and behaviours of different versions have certain differences.
- The client of ODPS is a java program and should be running in JRE environment. Please download and install JRE 1.6.
- If you want to quickly understand the use method of client, please refer to the client introduction in *ODPS Quick Start Manual.*

6.1.1.2 Command Summary

ODPS provides some operations for multiple objects, such as project, table, resource, function and so on. It also provides multiple service operations like SQL, MapReduce and security. The use habits of different functions (operations) are different. For example:

- For the operations of projects and tables, people are more accustomed to UNIX command format.
- ODPS SQL nestles up against SQL standard.
- MapReduce is used by ‘jar’ commands.

In order to provide different style command formats to distinguish all kinds of commands, we put forward the concept of command space (Shell). Certain functions can only be executed in a special shell:

- SQL Shell: User inputs the keyword ‘sql’ on the client and the client will consider the subsequent commands to be SQL commands. It does not support the commands of other shells until user inputs ‘quit’ to quit current shell or be switched to other shells.
- Security Shell: User inputs the keyword ‘security’ in the client and the client will consider the subsequent commands to be security commands. It does not support the commands of other shells until user inputs ‘quit’ to quit current shell or be switched to other shells.

ODPS Common commands and ‘jar’ command of MapReduce do not belong to any shell. They can be still running in SQL shell but can not be running in Security shell.

Use Examples:

```
odps:my_project> desc project --get current project information, can be executed successfully
odps:my_project> select * from my_table; --You have not entered sql shell and the command can not be executed.
odps:my_project> sql    --enter sql shell, can run SQL command and Common commands.
odps:sql:my_project> select * from my_table; --In sql shell, SQL statement can be executed successfully
odps:sql:my_project> desc my_table --Common commands can still be executed in sql shell.
```

Next gives the list of Common commands:

Commands	Description
help	
use <project_name>	
ls	
desc	
create	Create table statement is considered as SQL statement.
del	
kill	
status	
attach	
jar	Run MapReduce job.
log	View the log after running task
sql	Enter sql shell.

security	Enter security shell
quit	Quit current shell.

6.1.2 Configuration and Parameter Description

6.1.2.1 View Help Information

Use: view ‘help’ information.

Use method: you can view the ‘help’ information using ‘--help’ or ‘-h’ option. The command format is shows as follows:

```
odps --help
odps -h
```

Example:

```
$./bin/odps --help

Usage: odps [OPTIONS] <COMMAND> [OPTIONS] [ARGS]

Options:
    -p, --project                  Specify the default project
    -V, --version                  Print prodcut version
    -h, --help                     Print this help messages
    -c, --conf <CONFIG_FILE>       Specify a config file
    -f, --file <SCRIPT_FILE>       Run a script file

Available commands:
    help (h)       --Print this help messages or for help on a specific command

    use <PROJECT>   --Change default project in interactive shell

    ls       --List objects, such as tables,resource,functions,instances
    desc                     --Describe object meta
    create                   --Create an object
    del                      --Delete an object

    kill <INSTANCE_ID>     --Kill instance
    status <INSTANCE_ID>      Show instance status
    wait <INSTANCE_ID>        Wait for instance terminated

    sql                          Run a SQL query or active SQL shell

    jar                          Run Mapreduce job
```

security	Run security query
quit	Quit current shell

6.1.2.2 Specify Configuration File

Use: use the configuration file specified by user. If it is not specified, use the default configuration file ‘conf/odps.conf’.

Example:

```
./bin/odps -c your_conf.conf
```

6.1.2.3 EndPoint Configuration

Use: specify the service address of ODPS.

Use method: add the configuration for ‘endpoint’ in the configuration file ‘odps.conf’.

```
end_point=http://service.odps.aliyun.com/api
 # Public service address of ODPS
end_point=http://odps-ext.aliyun-inc.com/api
# Aliyun intranet service address of ODPS
```

🕮 **Note:**

- The selection of service address will affect the charge of data upload and download.
- ODPS also accepts HTTPS protocol.

6.1.2.4 Project Configuration

Use: enter the specified project.

Use method: add the configuration for ‘project_project’ in odps.conf. For example:

```
project_name=my_project
```

6.1.2.5 Configure ACCESS_ID and ACCESS_KEY

Use: specify user’s access_id and access_key.

Use method: add access_id and access_key in ‘odps.conf’, as shown below:

```
access_id=<access_id>
access_key=<access_key>
```

6.1.2.6 Specify Command Execution Script

Use: execute ODPS commands in script according to the order.

Use method: specify the command file name through the option ‘-f’. Such as:

```
odps -f <file_name>
```

6.1.2.7 View Version

Use: view the version information of ODPS client. Please note that this version is not ODPS service version.

Use method: specify ‘- -version’ or ‘-V’ in command line, such as:

```
odps --version
odps -V
```

Example:

```
$./bin/odps --version
ODPS Command-line Tools, version 0.12.0.
```

6.2 Eclipse Plug-in

6.2.1 Installation

In order to make user use Java SDK of MapReduce and UDF to do development work conveniently, ODPS provides Eclipse plug-in. This plug-in can simulate the running process of MapReduce and UDF, provide local debugging methods for users and provide simple template generation function.

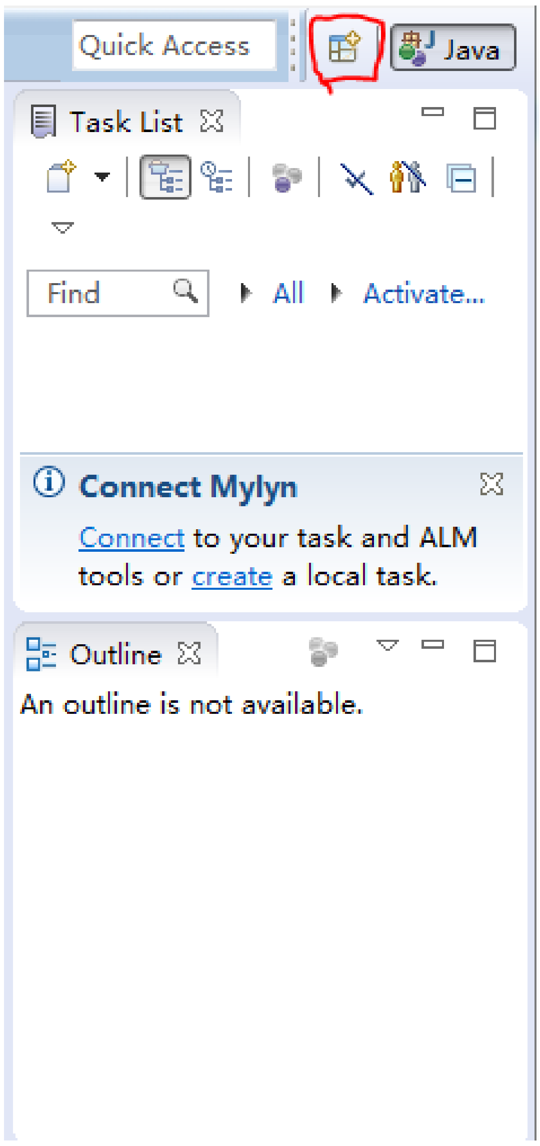
🕮 **Note:**

- Differently from local operation mode provided by MapReduce, Eclipse plug-in can not synchronize data with ODPS. User’s data needs to be copied into the directory ‘warehouse’ manually.

1) Afther downloading the Eclipse plug-in, decompress the software package and the following jar package will be shown:

```
odps-eclipse-plugin-bundle-0.15.0.jar
```

2) Put the plug-in into the subdirectory ‘plugins’ of Eclipse installation directory. Open Eclipse and click on <Open Perspective> at the right-upper corner:

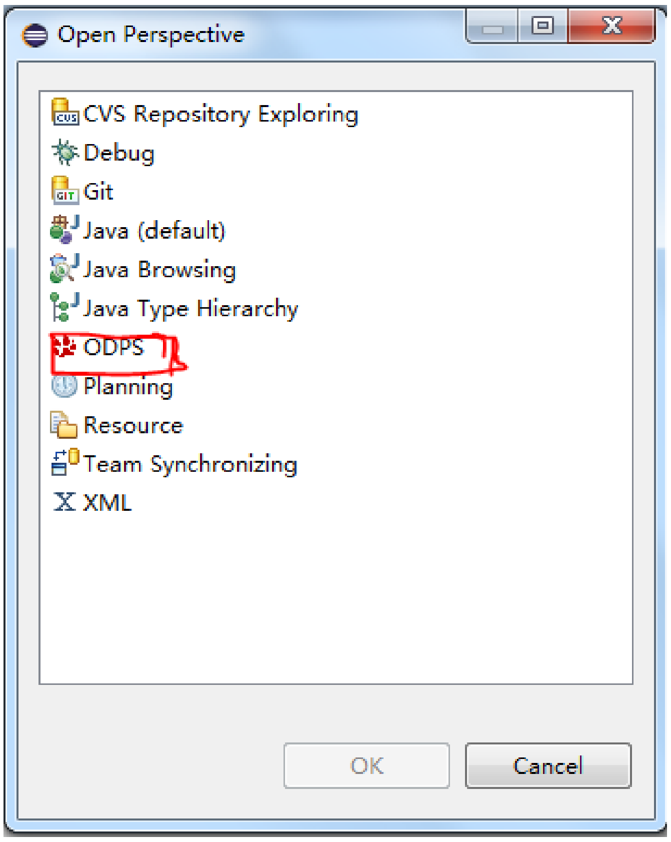


Connect Mylyn

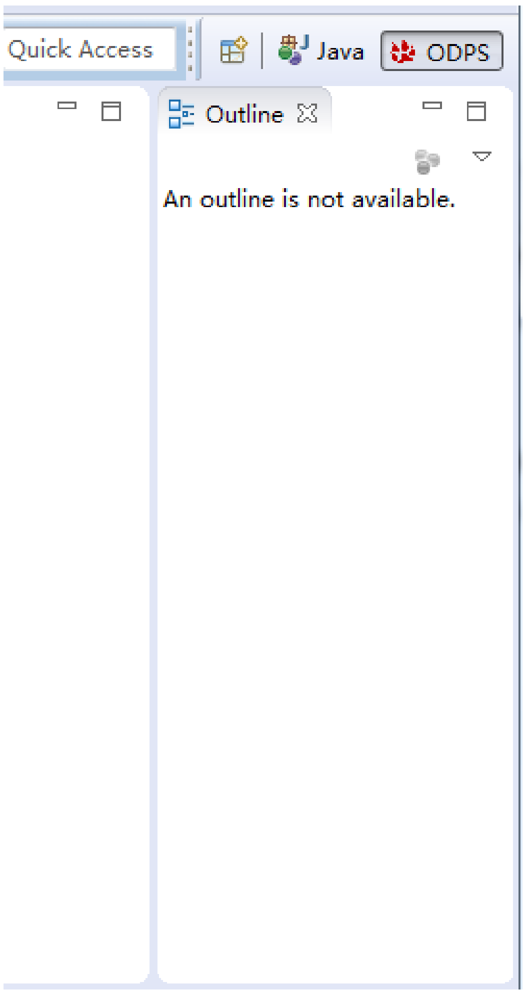
Connect to your task and ALM tools or create a local task.

An outline is not available.

3) Afther clicking, a dialog box pops up, shown as follows:



4) Selct ‘ODPS’ and click on <OK>. The icon ‘ODPS’ will be displayed at the right-upper corner, which indicates the pulg-in takes effect:



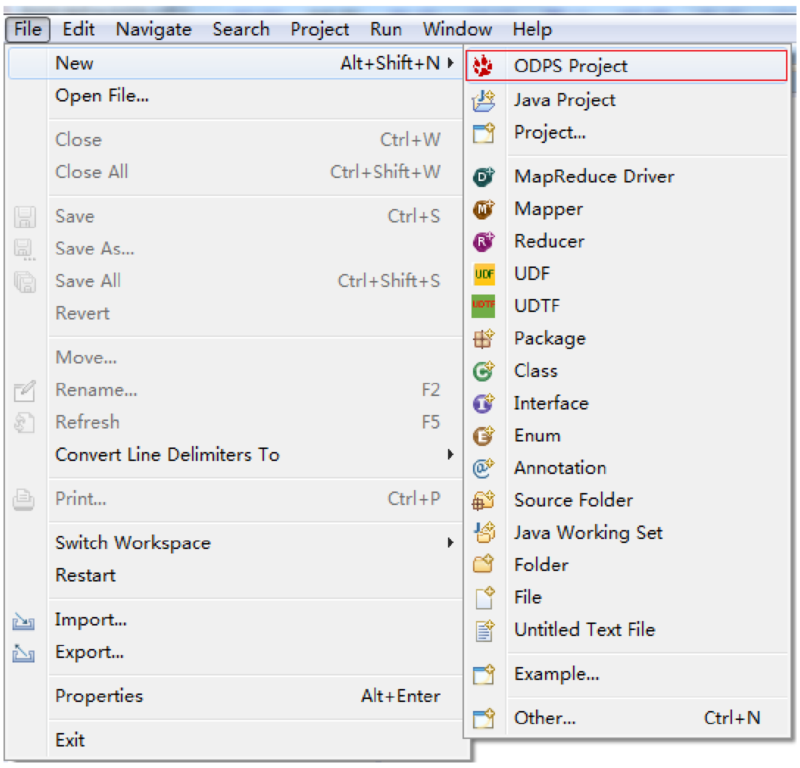
An outline is not available.

6.2.2 Create ODPS Project

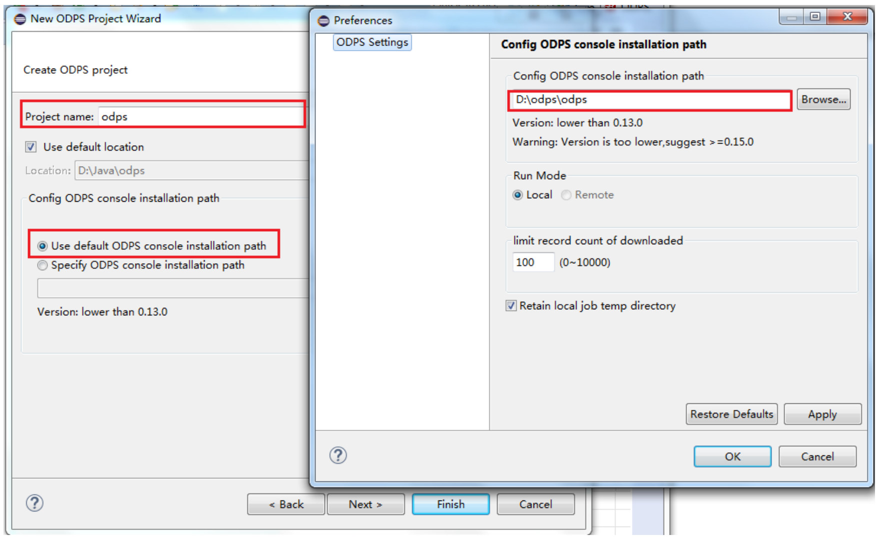
There are two methods to create ODPS project:

6.2.2.1 Method 1

1) Slect [File-> New-> Project-> ODPS->ODPS Project] to create the project (in the example, use 'ODPS' as the peoject name):



2) After creating ODPS project, the following dialog box will be popped up. Input Project name, and select the path of ODPS client. (The client should be uploaded at first.) At last, click <Finish>.


New ODPS Project Wizard

Create ODPS project

Project name: odps

[✓] Use default location

Location: D:\Java\odps

Config ODPS console installation path

[✓ Use default ODPS console installation path] Specify ODPS console installation path

Version: lower than 0.13.0

Preferences

ODPS Settings

Config ODPS console installation path

Config ODPS console installation path: D:\odps\odps

Version: lower than 0.13.0

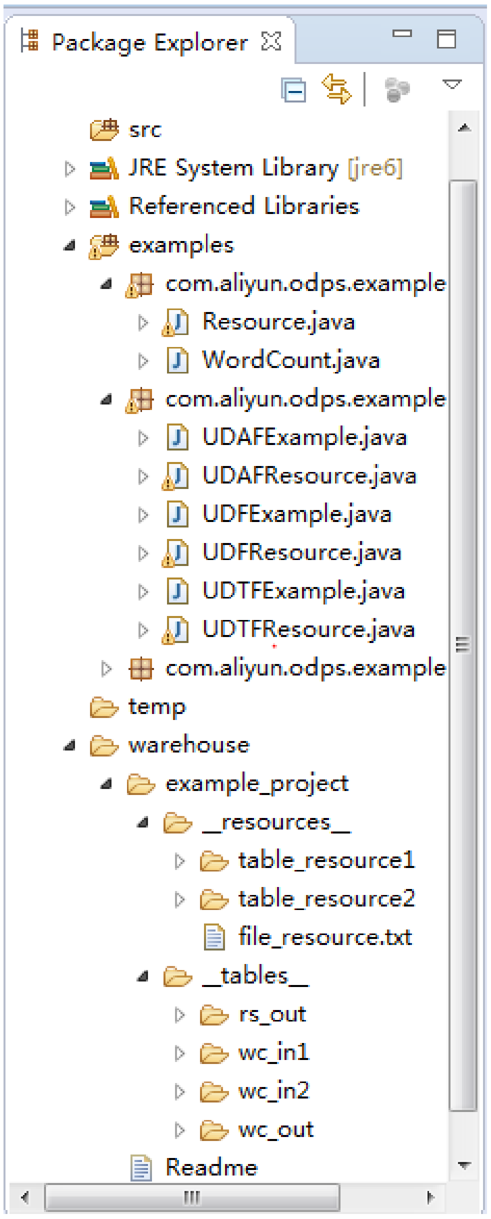
Warning: Version is too lower,suggest >=0.15.0

Run Mode: [✓ Local] Remote

limit record count of downloaded: 100 (0~10000)

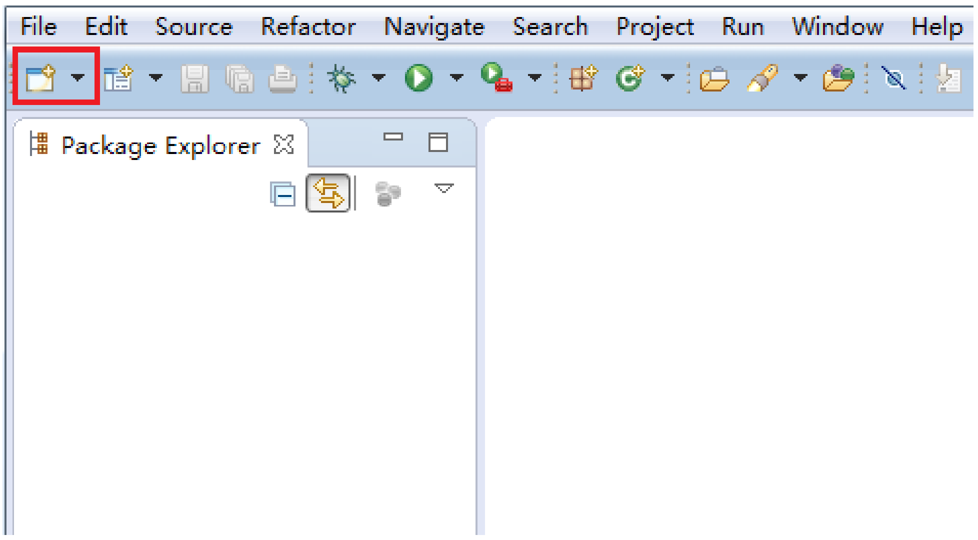
[✓] Retain local job temp directory

3) After creating project is finished, the following directory structure will be viewed in the left 'Package Explorer':

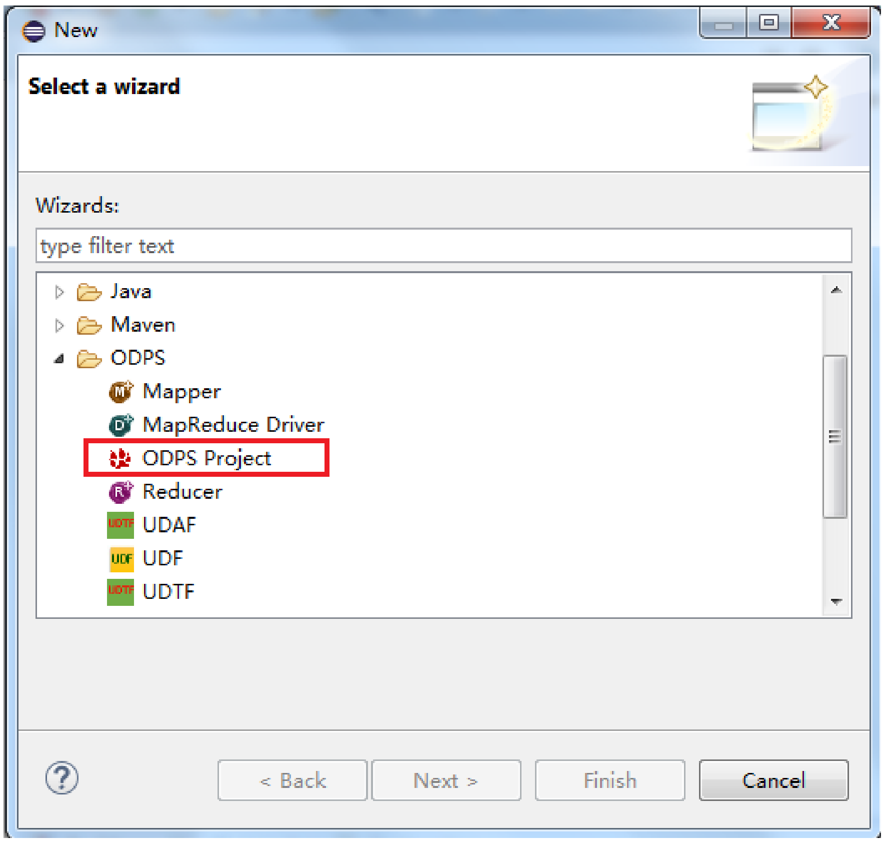


6.2.2.2 Method 2

1) Click <New> at the left-upper corner:


File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer

2) After the dialog box is popped up, select ‘ODPS Project’ and click on <Next>:



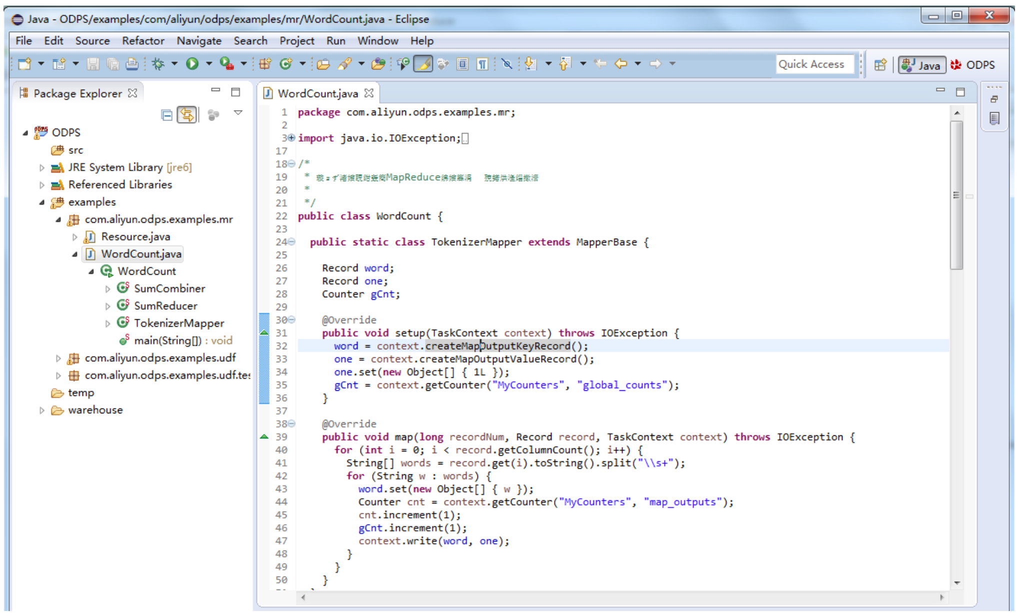
3) The subsequent operations are similar with Method 1.

The installation of ODPS Eclipse plug-in is completed. User can use this plug-in to write MapReduce or UDF programs.

6.2.3 MapReduce Development Plug-in Introduction

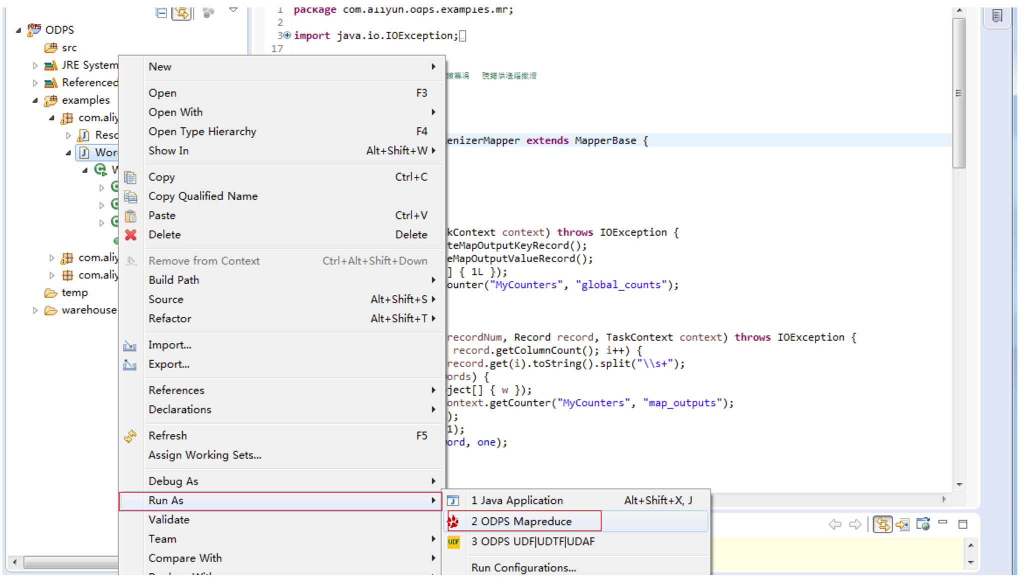
6.2.3.1 Run WordCount Example Quickly

1) Select WordCount example in OPDS project:

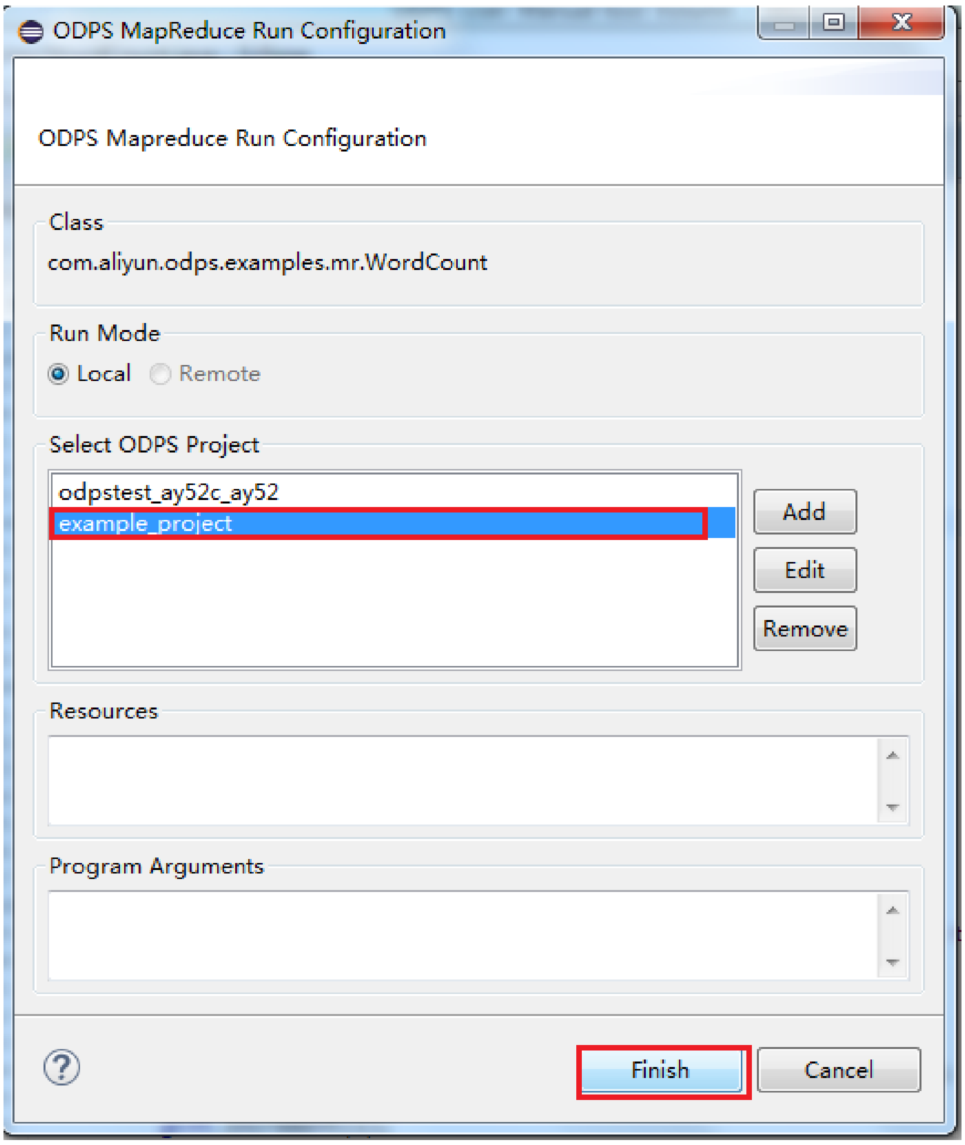

Java - ODPS/examples/com/aliyun/odps/examples/mr/WordCount.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
ODPS
Package Explorer
ODPS
src
JRE System Library [jre6]
Referenced Libraries
examples
com.aliyun.odps.examples.mr
Resource.java
WordCount.java
WordCount
SumCombiner
SumReducer
TokenizerMapper
main(String[]) : void
com.aliyun.odps.examples.udf
com.aliyun.odps.examples.udf.tes
temp
warehouse
WordCount.java

```
package com.aliyun.odps.examples.mr;

import java.io.IOException;

/*
 * [illegible]MapReduce[illegible]
 *
 */
public class WordCount {

  public static class TokenizerMapper extends MapperBase {

    Record word;
    Record one;
    Counter gCnt;

    @Override
    public void setup(TaskContext context) throws IOException {
      word = context.createMapOutputKeyRecord();
      one = context.createMapOutputValueRecord();
      one.set(new Object[] { 1L });
      gCnt = context.getCounter("MyCounters", "global_counts");
    }

    @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.set(new Object[] { w });
          Counter cnt = context.getCounter("MyCounters", "map_outputs");
          cnt.increment(1);
          gCnt.increment(1);
          context.write(word, one);
        }
      }
    }
```

2) Right-click on ‘WordCount.java’ and click <Run As-> ODPS MapReduce>, as follows:



3) After the dialog box is popped up, select ‘example_project’ and click on <Finish>:



ODPS MapReduce Run Configuration

ODPS Mapreduce Run Configuration

Class

com.aliyun.odps.examples.mr.WordCount

Run Mode

[✓ Local] Remote

Select ODPS Project

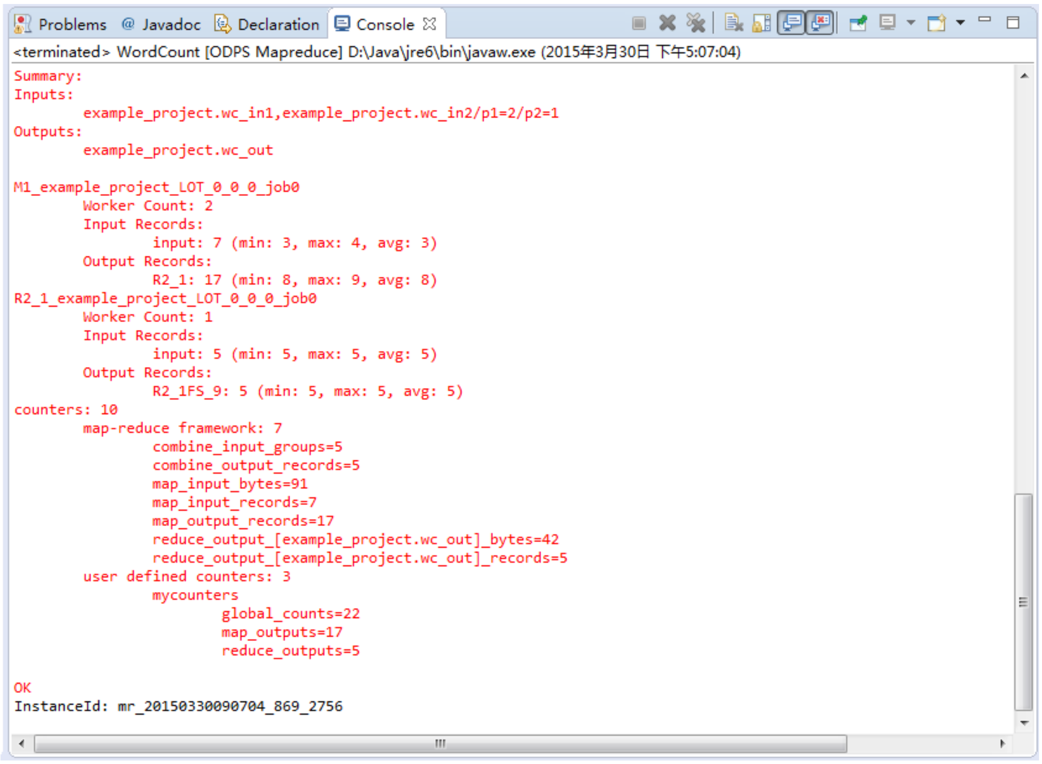
odpstest_ay52c_ay52

example_project

Resources

Program Arguments

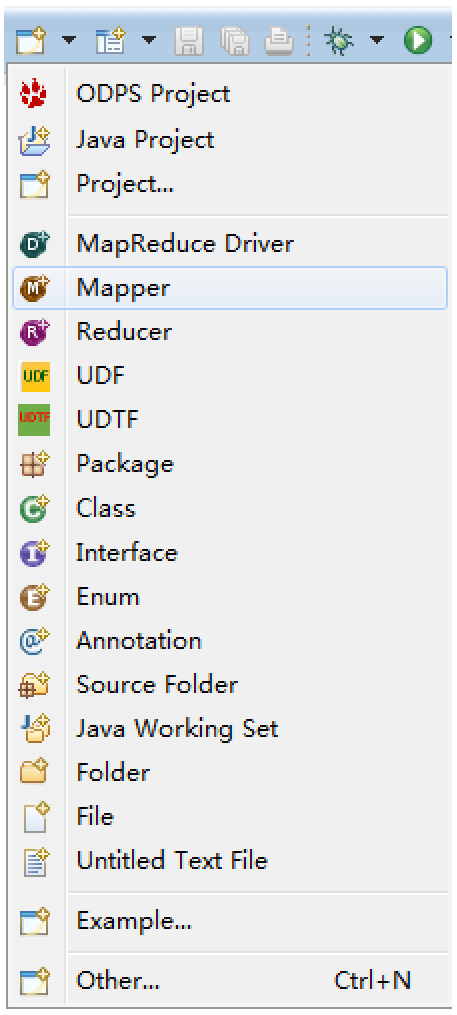
4) After running is successful, the following result will be displayed:



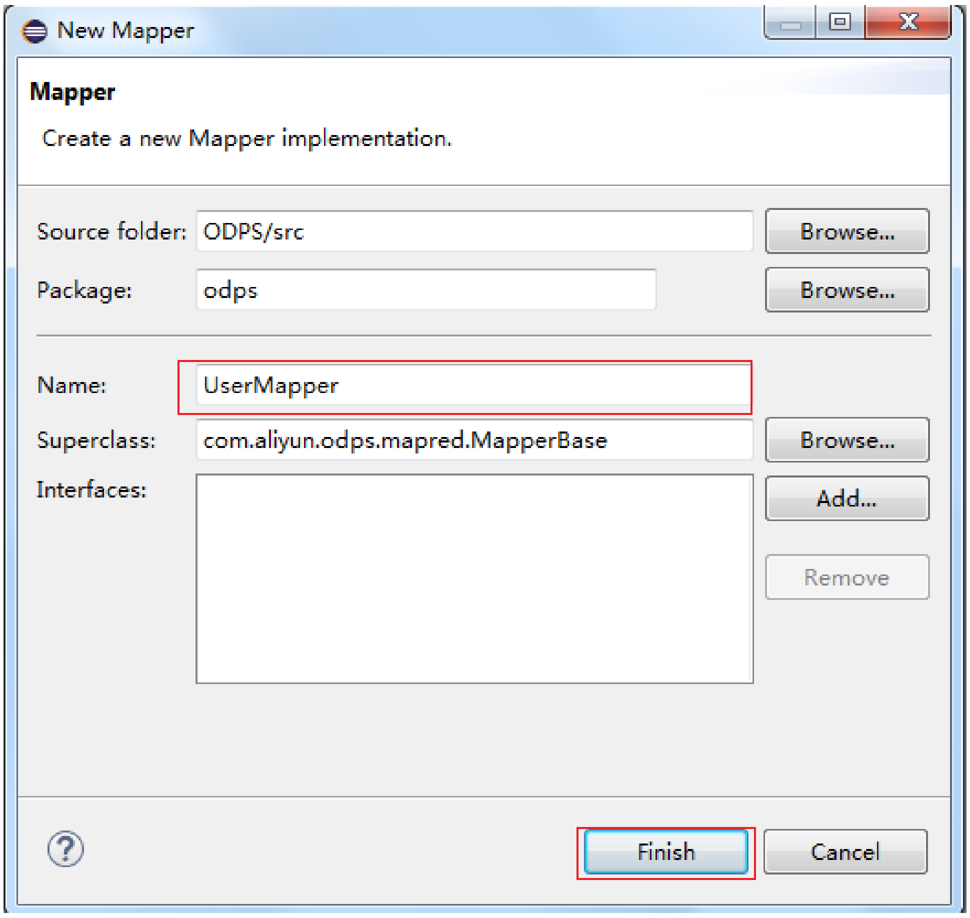
```
<terminated> WordCount [ODPS Mapreduce] D:\Java\jre6\bin\javaw.exe (2015年3月30日 下午5:07:04)
Summary:
Inputs:
        example_project.wc_in1,example_project.wc_in2/p1=2/p2=1
Outputs:
        example_project.wc_out

M1_example_project_LOT_0_0_0_job0
        Worker Count: 2
        Input Records:
                input: 7 (min: 3, max: 4, avg: 3)
        Output Records:
                R2_1: 17 (min: 8, max: 9, avg: 8)
R2_1_example_project_LOT_0_0_0_job0
        Worker Count: 1
        Input Records:
                input: 5 (min: 5, max: 5, avg: 5)
        Output Records:
                R2_1FS_9: 5 (min: 5, max: 5, avg: 5)
counters: 10
        map-reduce framework: 7
                combine_input_groups=5
                combine_output_records=5
                map_input_bytes=91
                map_input_records=7
                map_output_records=17
                reduce_output_[example_project.wc_out]_bytes=42
                reduce_output_[example_project.wc_out]_records=5
        user defined counters: 3
                mycounters
                        global_counts=22
                        map_outputs=17
                        reduce_outputs=5

OK
InstanceId: mr_20150330090704_869_2756
```

6.2.3.2 Run Uer-defined MapReduce Program

1) Right-click on ‘src’ directory. Select <New -> Mapper>:



2) After selecting ‘Mapper’ and the following dialog box is displayed. Input the name of Mapper class and click on <Finish>:



Mapper

Create a new Mapper implementation.

Source folder: ODPS/src

Package: odps

Name: UserMapper

Superclass: com.aliyun.odps.mapred.MapperBase

Interfaces:

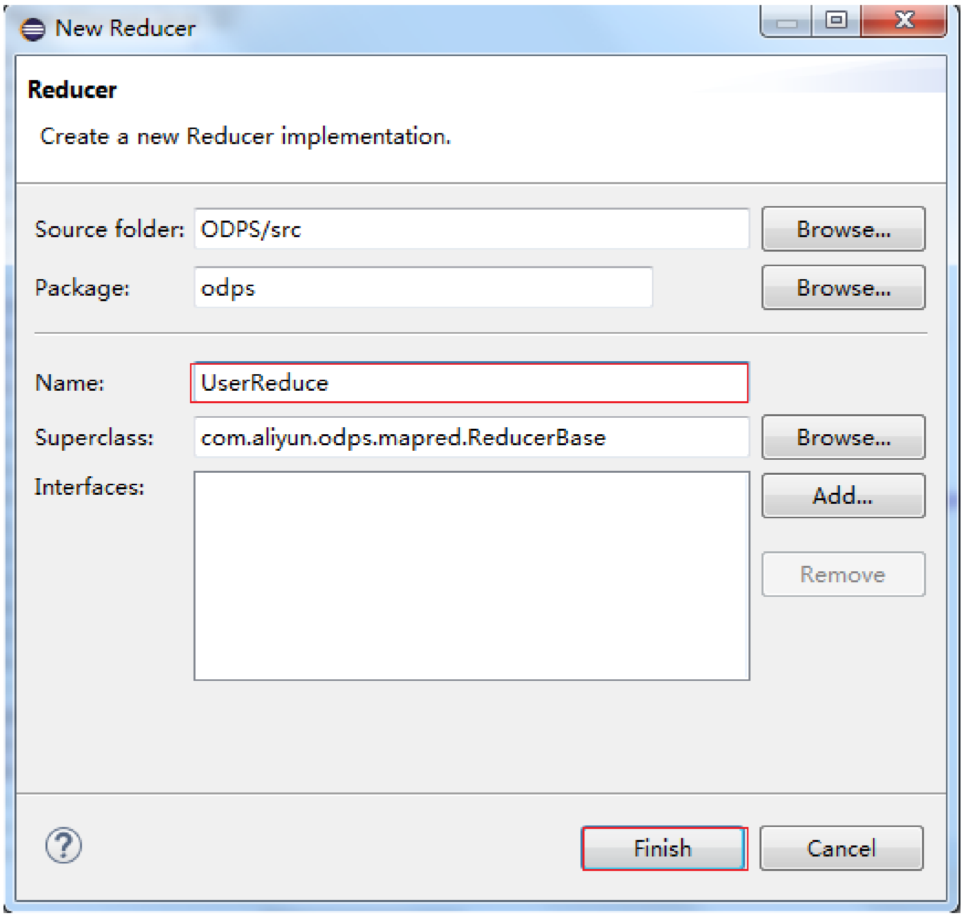
3) Now you can find a file ‘UserMapper.java’ is generated in the directory ‘src’ in ‘Package Explorer’. The content of this file is a template of Mapper class:

```
package odps;

import java.io.IOException;

import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.MapperBase;

public class UserMapper extends MapperBase {

    @Override
    public void setup(TaskContext context) throws IOException {
    }

    @Override
    public void map(long recordNum, Record record, TaskContext context)
            throws IOException {
    }

    @Override
    public void cleanup(TaskContext context) throws IOException {
    }

}
```

4) In the template, the configured package name defaults to 'odps'. You can modify it according to your actual requirement. Write the template contents as follows:

```
package odps;

import java.io.IOException;

import com.aliyun.odps.counter.Counter;
import com.aliyun.odps.data.Record;
import com.aliyun.odps.mapred.MapperBase;

public class UserMapper extends MapperBase {

    Record word;
    Record one;
    Counter gCnt;

    @Override
    public void setup(TaskContext context) throws IOException {
          word = context.createMapOutputKeyRecord();
          one = context.createMapOutputValueRecord();
          one.set(new Object[] { 1L });
          gCnt = context.getCounter("MyCounters", "global_counts");
    }

    @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.set(new Object[] { w });
                Counter cnt = context.getCounter("MyCounters", "map_outputs");
                cnt.increment(1);
                gCnt.increment(1);
                context.write(word, one);
              }
            }
          }

    @Override
    public void cleanup(TaskContext context) throws IOException {
    }

}
```

5) In the same method, right-click on ‘src’ directory and select <New -> Reduce>:


New Reducer

Reducer

Create a new Reducer implementation.

Source folder: ODPS/src

Package: odps

Name: UserReduce

Superclass: com.aliyun.odps.mapred.ReducerBase

Interfaces:

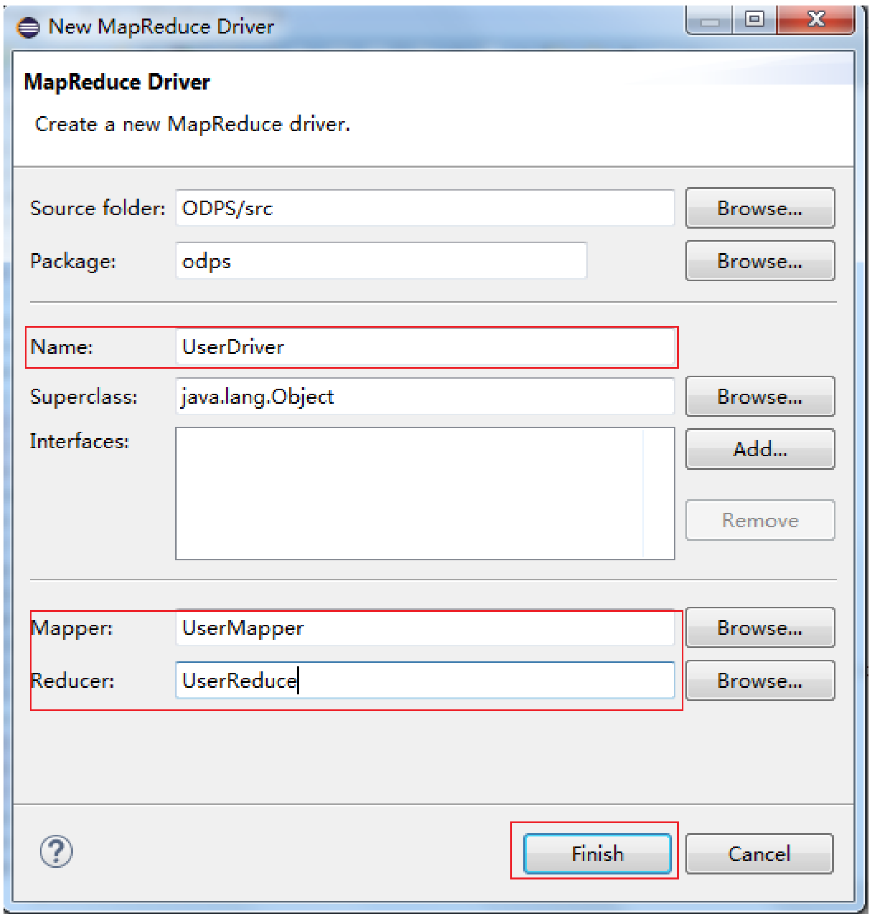
Input the name of Reduce class. (In this example, use ‘UserReduce’ as the class name):

6) In ‘Package Explorer’, a file name ‘UseReduce.java’ is generated in the directory ‘src’. This file content is a template of Reduce class. Edit the template:

```
package odps;


import java.io.IOException;

import java.util.Iterator;


import com.aliyun.odps.counter.Counter;

import com.aliyun.odps.data.Record;

import com.aliyun.odps.mapred.ReducerBase;


public class UserReduce extends ReducerBase {


    private Record result;

    Counter gCnt;


    @Override

    public void setup(TaskContext context) throws IOException {

          result = context.createOutputRecord();

          gCnt = context.getCounter("MyCounters", "global_counts");

    }


    @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);

          Counter cnt = context.getCounter("MyCounters", "reduce_outputs");
```

```
                cnt.increment(1);
                gCnt.increment(1);

                context.write(result);
            }

        @Override
        public void cleanup(TaskContext context) throws IOException {
        }

    }
```

7) Create 'main' function: right-click on 'src' and select <New -> MapReduce Driver>. Fill in Driver Name (in this example, use 'UserDriver' as the name), Mapper and Recduce (in this example use 'UserMapper' and 'UserReduce' as corresponding name) and click on <Finish>. The file 'MyDriver.java file' is also displayed in 'src' directory:


New MapReduce Driver

MapReduce Driver

Create a new MapReduce driver.

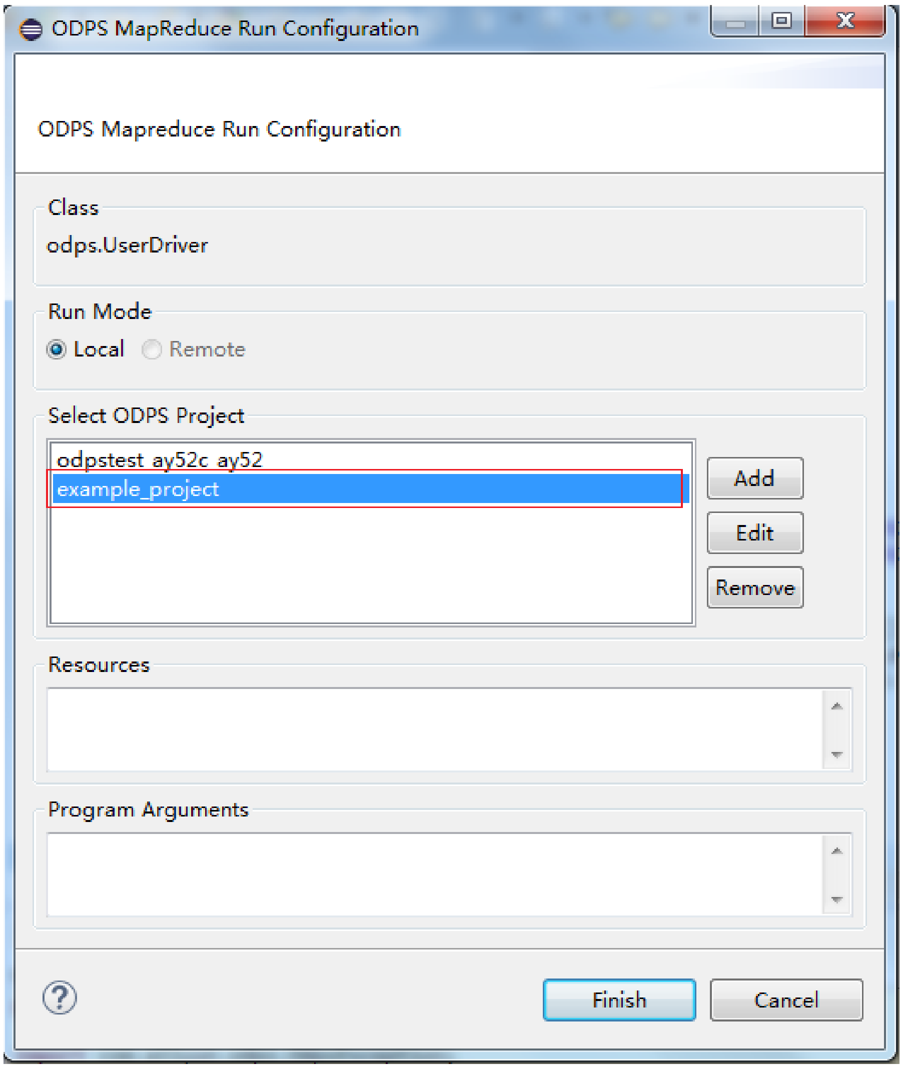
Source folder: ODPS/src
Package: odps
Name: UserDriver
Superclass: java.lang.Object
Interfaces:
Mapper: UserMapper
Reducer: UserReduce

8) Edit the contents of driver:

```
package odps;

import com.aliyun.odps.OdpsException;
import com.aliyun.odps.data.TableInfo;
import com.aliyun.odps.examples.mr.WordCount.SumCombiner;
import com.aliyun.odps.examples.mr.WordCount.SumReducer;
import com.aliyun.odps.examples.mr.WordCount.TokenizerMapper;
import com.aliyun.odps.mapred.JobClient;
import com.aliyun.odps.mapred.RunningJob;
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 UserDriver {

    public static void main(String[] args) throws OdpsException {
        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("wc_in1").cols(new String[] { "col2",
"col3" }).build(), job);

InputUtils.addTable(TableInfo.builder().tableName("wc_in2").partSpec("p1=2/p2=1").bui
ld(), job);
        OutputUtils.addTable(TableInfo.builder().tableName("wc_out").build(), job);
```

```
            RunningJob rj = JobClient.runJob(job);

            rj.waitForCompletion();

        }


    }
```

9) Run MapReduce program. Right-click on 'UserDriver.java' and select <Run As -> ODPS MapReduce> to pop up the following dialog box:


ODPS MapReduce Run Configuration

ODPS Mapreduce Run Configuration

Class
odps.UserDriver

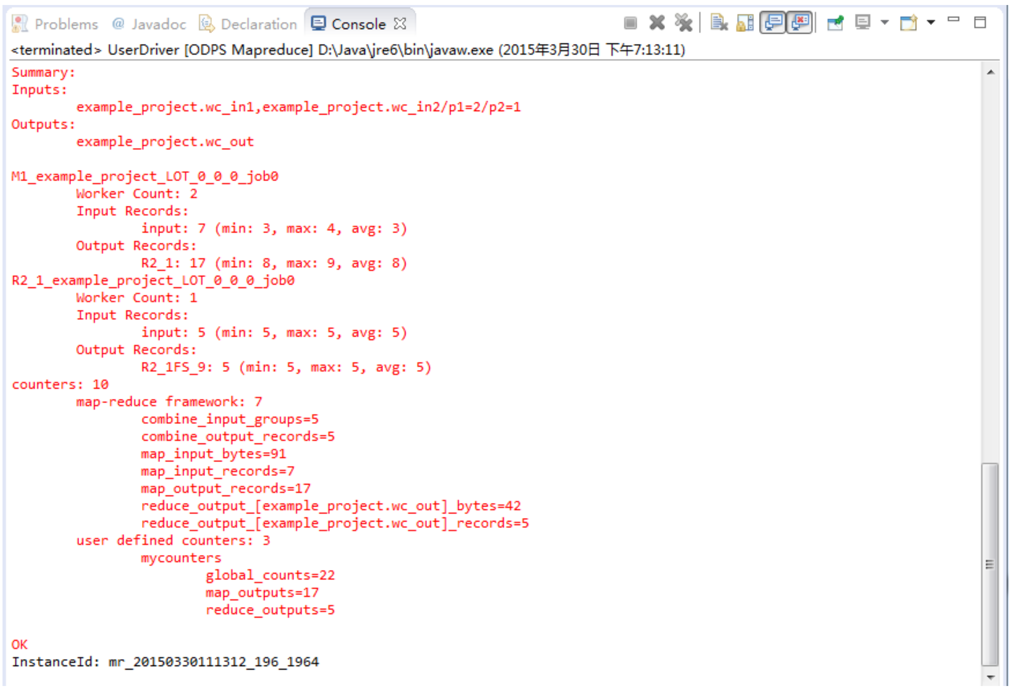
Run Mode
[✓ Local] Remote

Select ODPS Project
odpstest_ay52c_ay52
[✓ example_project]

Resources

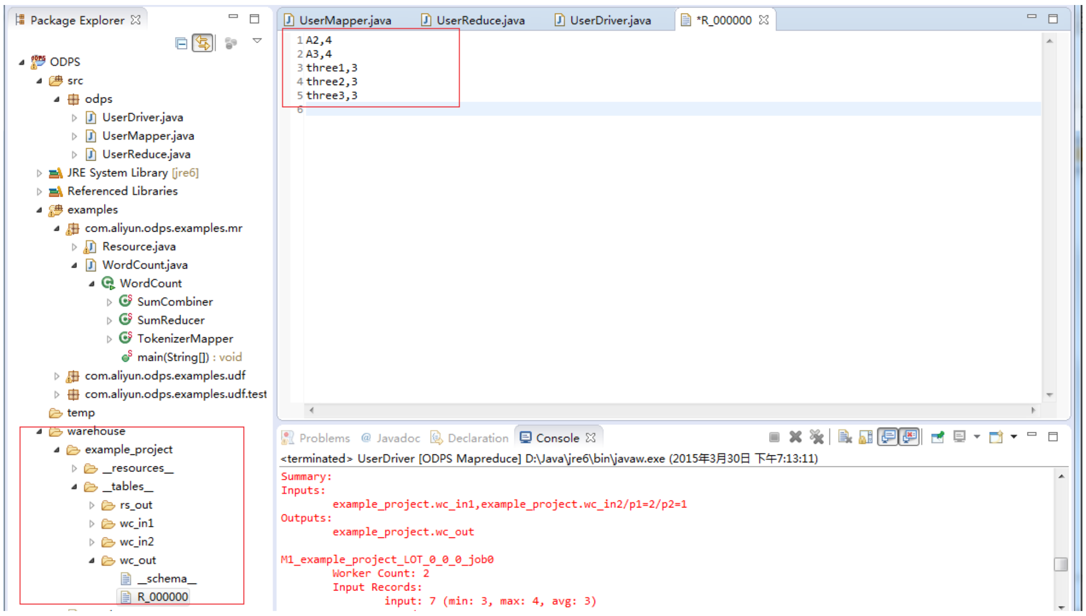
Program Arguments

10) Select 'example_project' as the ODPS Project and click on <Finish> to run MapReduce program on the local:



```
<terminated> UserDriver [ODPS Mapreduce] D:\Java\jre6\bin\javaw.exe (2015年3月30日 下午7:13:11)
Summary:
Inputs:
        example_project.wc_in1,example_project.wc_in2/p1=2/p2=1
Outputs:
        example_project.wc_out

M1_example_project_LOT_0_0_0_job0
        Worker Count: 2
        Input Records:
                input: 7 (min: 3, max: 4, avg: 3)
        Output Records:
                R2_1: 17 (min: 8, max: 9, avg: 8)
R2_1_example_project_LOT_0_0_0_job0
        Worker Count: 1
        Input Records:
                input: 5 (min: 5, max: 5, avg: 5)
        Output Records:
                R2_1FS_9: 5 (min: 5, max: 5, avg: 5)
counters: 10
        map-reduce framework: 7
                combine_input_groups=5
                combine_output_records=5
                map_input_bytes=91
                map_input_records=7
                map_output_records=17
                reduce_output_[example_project.wc_out]_bytes=42
                reduce_output_[example_project.wc_out]_records=5
        user defined counters: 3
                mycounters
                        global_counts=22
                        map_outputs=17
                        reduce_outputs=5

OK
InstanceId: mr_20150330111312_196_1964
```

11) If the information is output as mentioned above, it indicates that local operation runs successfully. The output result is saved in the directory ‘warehouse’. Refresh ODPS project:



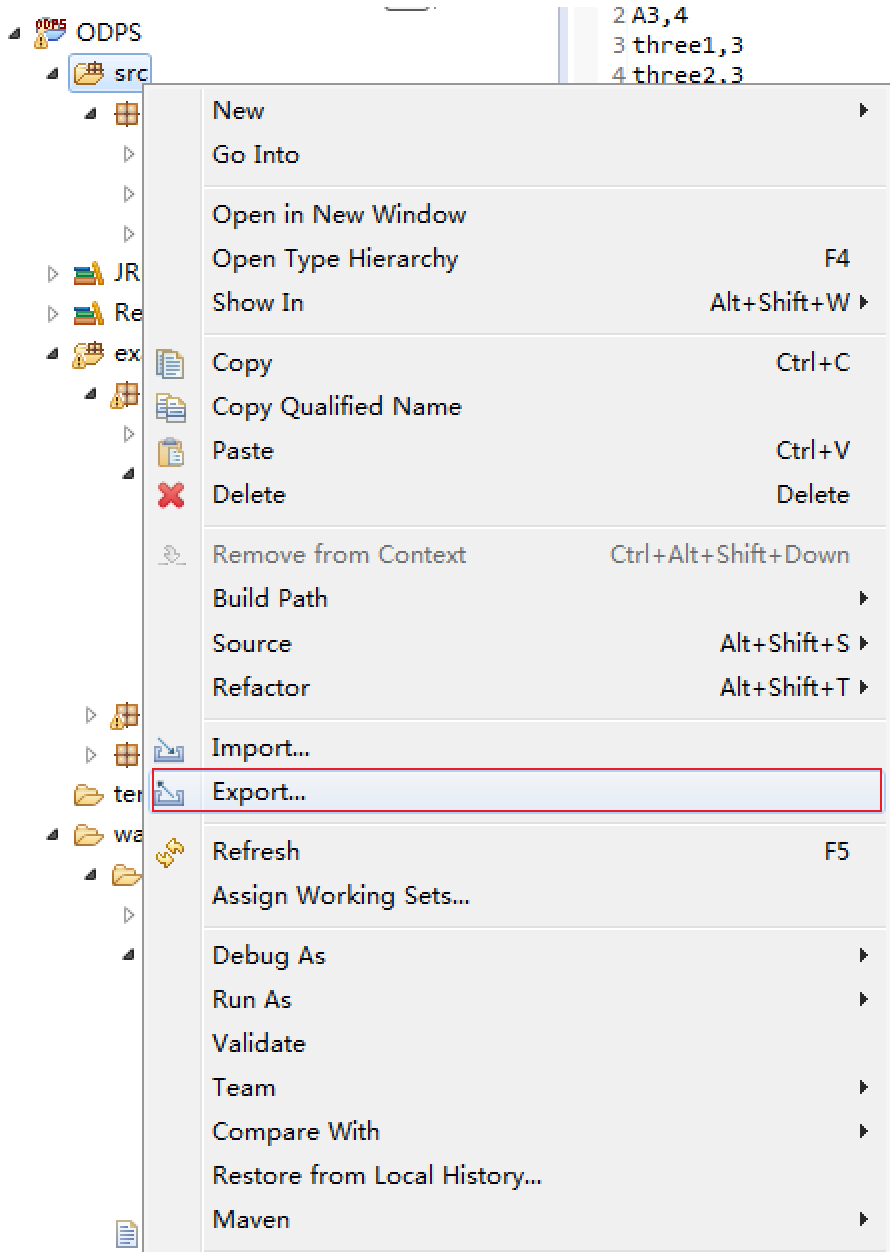
```
ODPS
  src
    odps
      UserDriver.java
      UserMapper.java
      UserReduce.java
  JRE System Library [jre6]
  Referenced Libraries
  examples
    com.aliyun.odps.examples.mr
      Resource.java
      WordCount.java
        WordCount
          SumCombiner
          SumReducer
          TokenizerMapper
          main(String[]) : void
    com.aliyun.odps.examples.udf
    com.aliyun.odps.examples.udf.test
  temp
  warehouse
    example_project
      __resources__
      __tables__
        rs_out
        wc_in1
        wc_in2
        wc_out
          __schema__
          R_000000
```

```
A2,4
A3,4
three1,3
three2,3
three3,3
```

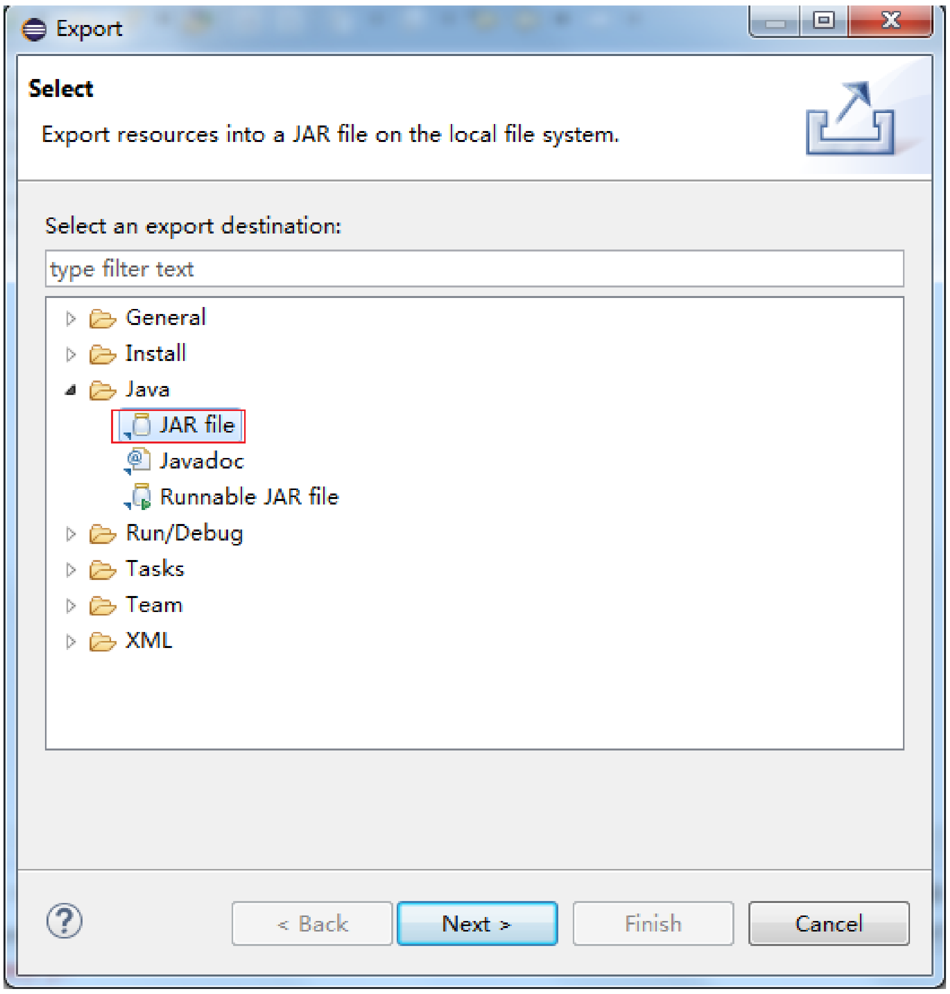
```
<terminated> UserDriver [ODPS Mapreduce] D:\Java\jre6\bin\javaw.exe (2015年3月30日 下午7:13:11)
Summary:
Inputs:
        example_project.wc_in1,example_project.wc_in2/p1=2/p2=1
Outputs:
        example_project.wc_out

M1_example_project_LOT_0_0_0_job0
        Worker Count: 2
        Input Records:
                input: 7 (min: 3, max: 4, avg: 3)
```

‘wc_out’ is the output directory and ‘R_000000’ is result file. Through local debugging, the result is confrmed to be correct and you can package MapReduce program through Eclipse export function. After it is packaged, upload the jar package to ODPS. About how to execute MapReduce in distributed environment, please refer to ODPS User Manual-Quick Start Volume.

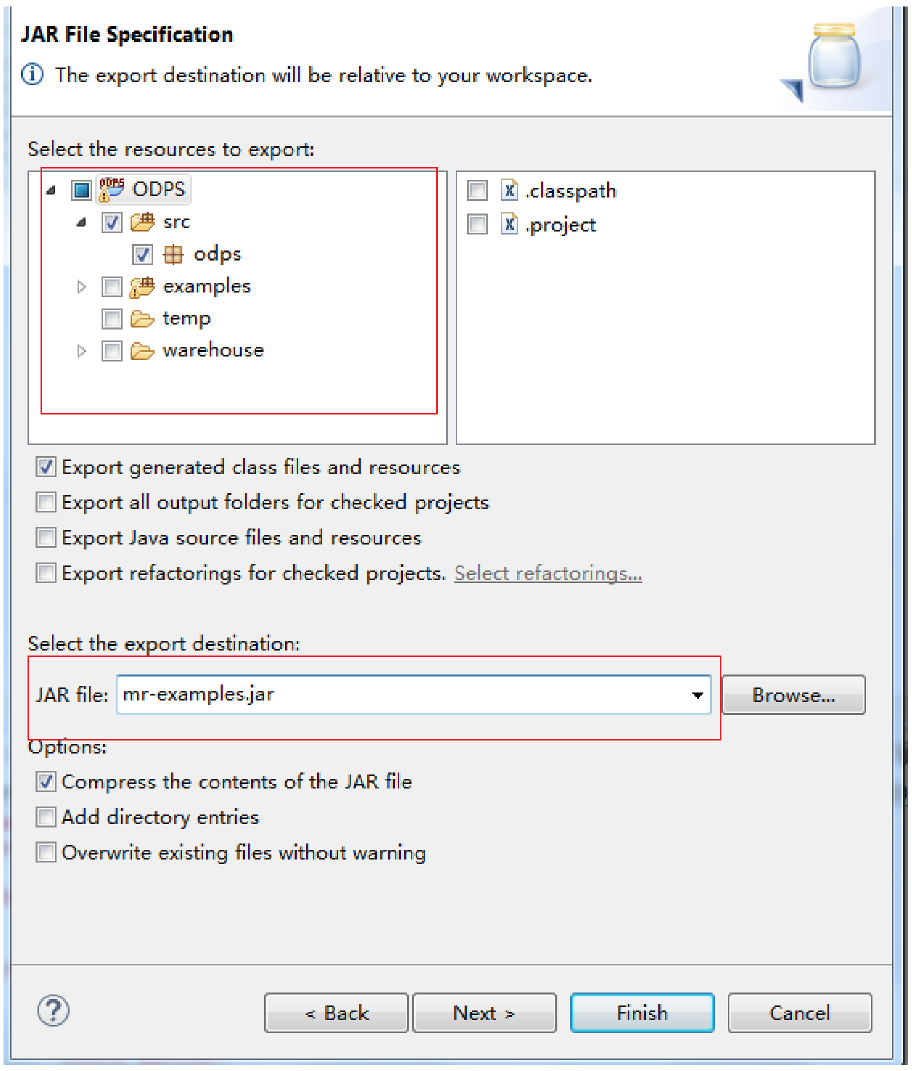
12) After the local debugging passed, user can package the codes into jar package through Eclipse Export function, provided for subsequent distributed environment. In this example, we nominate the package 'mr-examples.jar'. Select the directory 'src' and click on <Export>:



13) Select 'Jar File' as an export destination:



14) You just need to export the package in 'src'. Specify the name of Jar File to be 'mr-examples.jar':


JAR File Specification

The export destination will be relative to your workspace.

Select the resources to export:

- [■] ODPS
  - [✓] src
    - [✓] odps
  - [ ] examples
  - [ ] temp
  - [ ] warehouse
- [ ] .classpath
- [ ] .project

- [✓] Export generated class files and resources
- [ ] Export all output folders for checked projects
- [ ] Export Java source files and resources
- [ ] Export refactorings for checked projects. Select refactorings...

Select the export destination:

JAR file: mr-examples.jar

Options:

- [✓] Compress the contents of the JAR file
- [ ] Add directory entries
- [ ] Overwrite existing files without warning

15) Click on <Next> to export the jar file successfully.

If you want to simulate new Project creation in the local, you can create a subdirectory (has same level with example_project) in the directory ‘warehourse’. The directory hierarchy structure is shown as follows:

```
<warehouse>
     |____example_project (project directoty)
              |
              |___table_name1 (non-partition table)
              |         |____ data (file)
              |         |
              |         |____ <__schema__> (file)
              |
              | __table_name2 (partition table)
```

```
        |          |____ partition_name=partition_value (partition directory)
        |          |                |____ data (file)
        |          |
        |          |____ <__schema__> (file)
        |
        |__ <__resources__>
                 |
                 |
                 |___ file_resource_name (file resource)
```

Schema file example:

```
Non-partition table:
project=project_name
table=table_name
columns=col_name1:col_type,col_name2:col_type

partition table:
project=project_name
table=table_name
columns=col_name1:col_type,col_name2:col_type
partitions=col_name1:col_type,col_name2:col_type
```

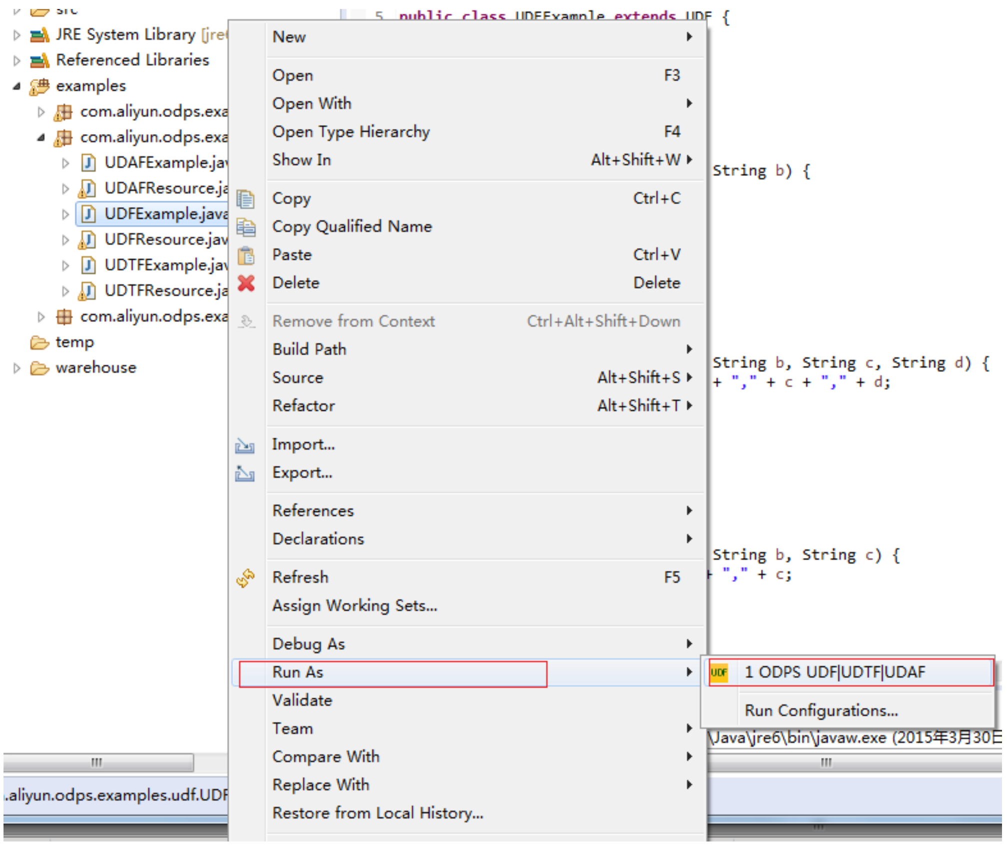
🕮 **Note:**

- If mapReduce program runs in the local, the default is to search corresponding tables or resources from the directory 'warehouse'. If the tables or resources are not existent, corresponding data will be downloaded from the server and saved in 'warehouse'. Then run MapReduce in the local.
- After running MapReduce is finished, please refresh the directoty 'warehouse' to view generated result.

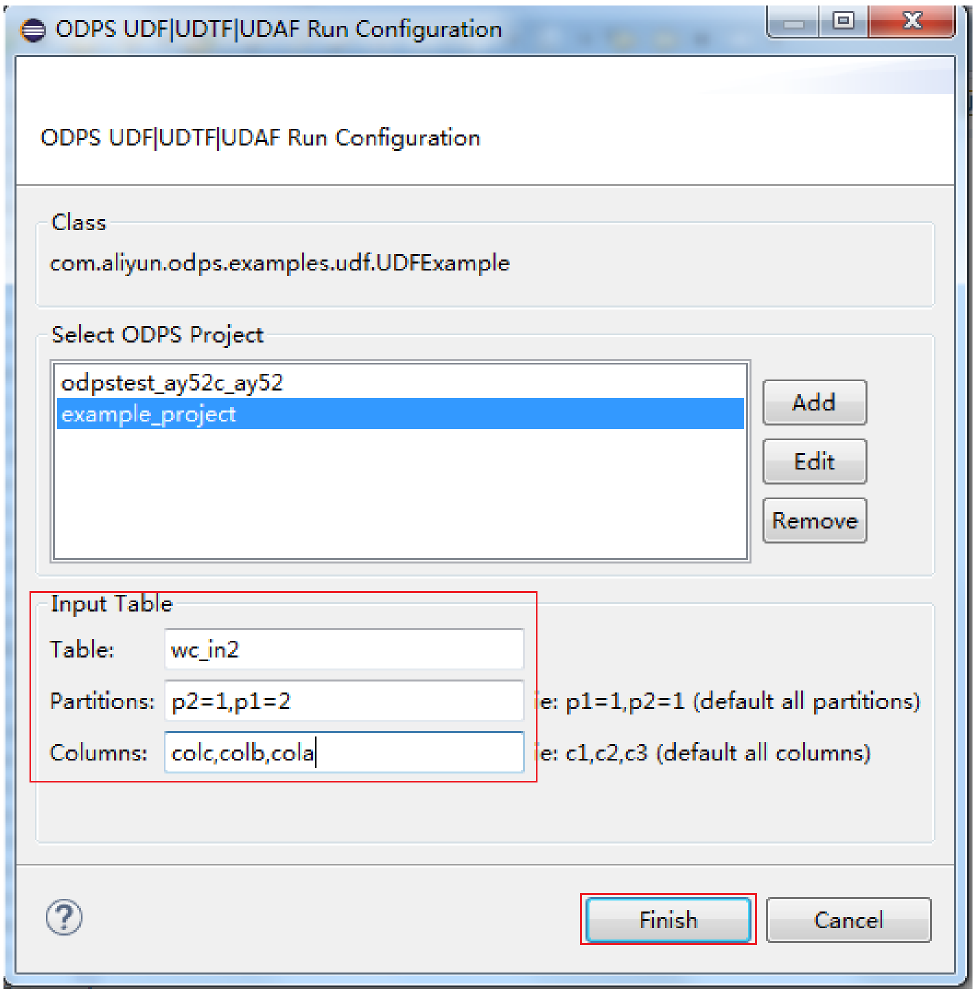
6.2.3.3 UDF Development Plug-in Introduction

6.2.3.3.1 Run UDF Example Quickly

1) Select 'UDFExample.java' and right-click on it to select <Run As -> Run UDF|UDAF|UDTF>:



2) Configure the input table information:


ODPS UDF|UDTF|UDAF Run Configuration

ODPS UDF|UDTF|UDAF Run Configuration

Class

com.aliyun.odps.examples.udf.UDFExample

Select ODPS Project

odpstest_ay52c_ay52
example_project

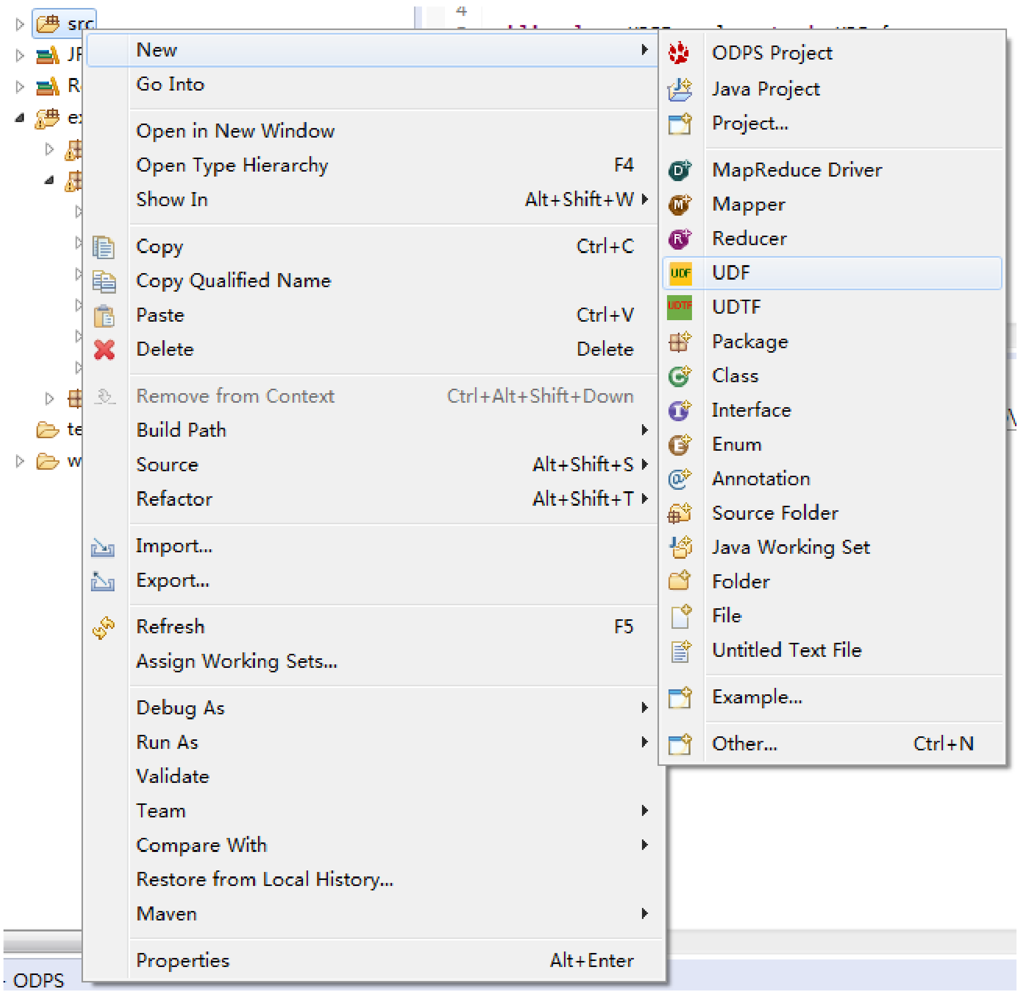
Input Table

Table:	wc_in2	
Partitions:	p2=1,p1=2	e: p1=1,p2=1 (default all partitions)
Columns:	colc,colb,cola	e: c1,c2,c3 (default all columns)

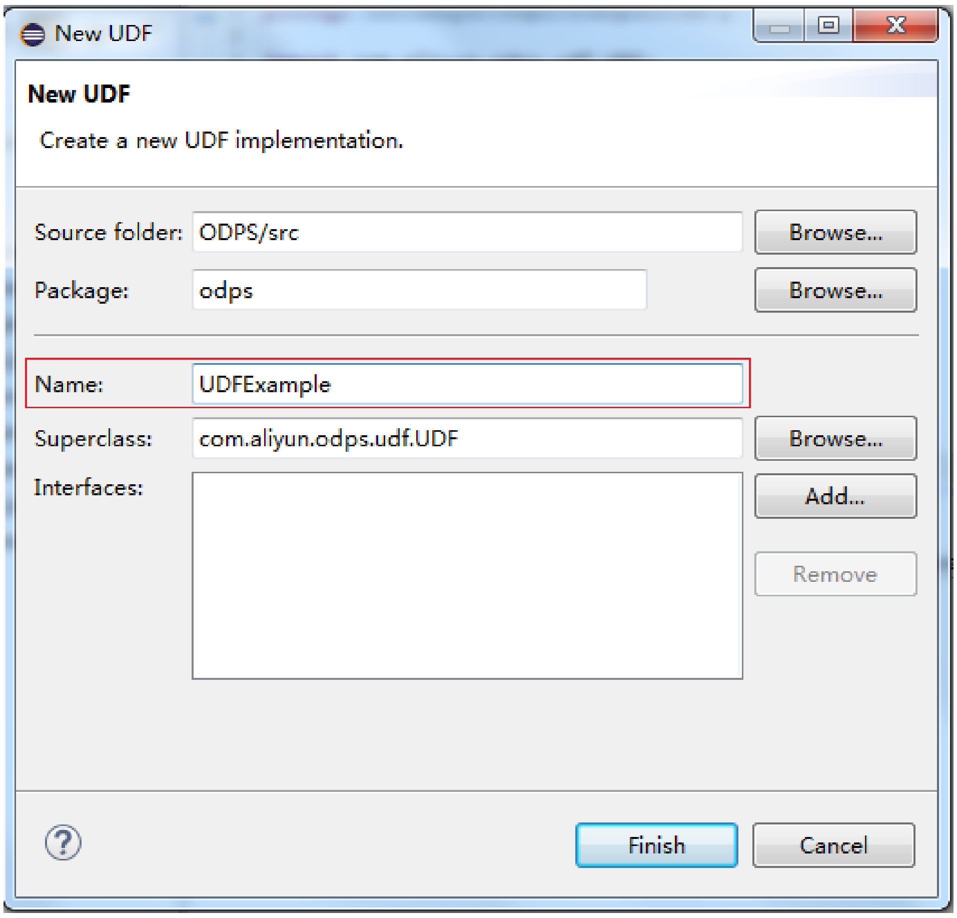
3) Click on <Finish> to run UDF and get output result. In the configuration mentioned above, ‘Table’ indicates the input table of UDF, ‘Partitions’ indicates reading data in specified partitions, separated by commas and ‘Columns’ indicates the columns, input as the parmaters of UDF in proper order, separated by commas.

6.2.3.3.2 Run User-defined UDF Program

1) Right-click on ‘src’ directory and select <New-> UDF>:



2) Appear the following dialog box. Fill in UDF class name and click on <Finish> to comfirm it.


New UDF

New UDF

Create a new UDF implementation.

Source folder: ODPS/src

Package: odps

Name: UDFExample

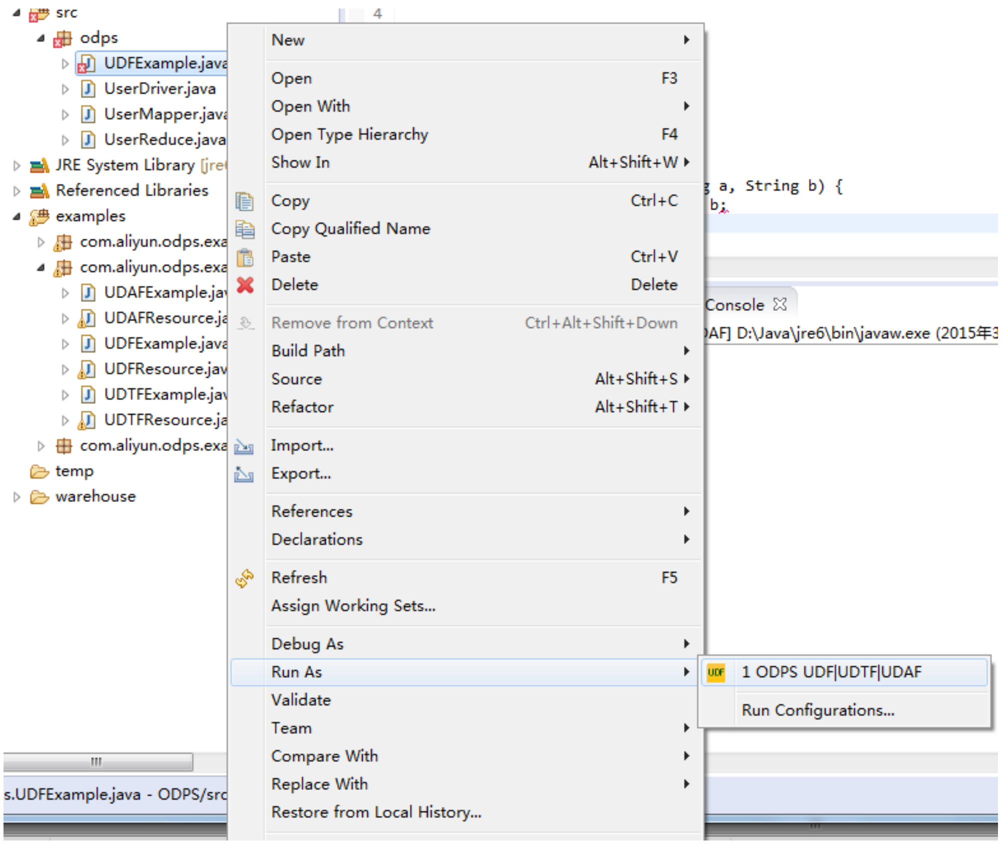
Superclass: com.aliyun.odps.udf.UDF

Interfaces:

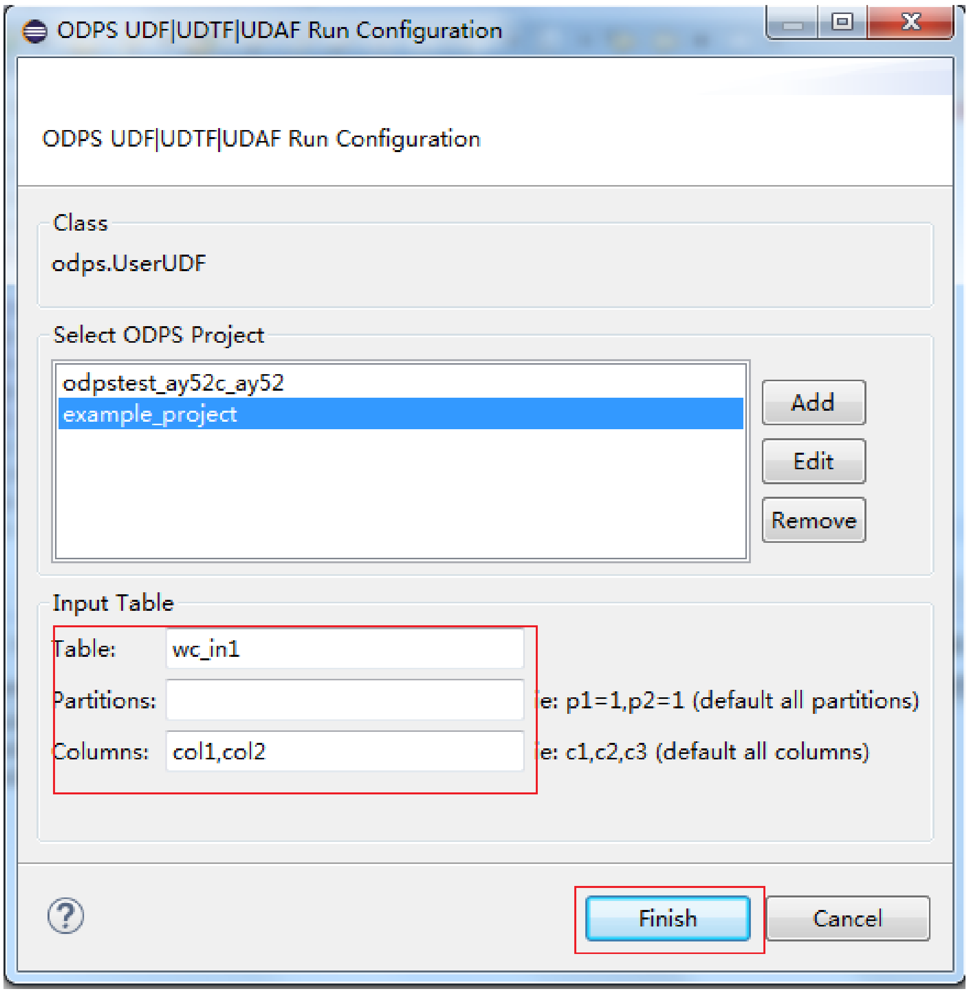
3) A same name Java file with UDF class is generated in ‘src’. Edit the file contents:

```
package odps;

import com.aliyun.odps.udf.UDF;

public class UserUDF extends UDF {

    /**
     * project: example_project
     * table: wc_in1
     * columns: col1,col2
     *
     */
    public String evaluate(String a, String b) {
      return "ss2s:" + a + "," + b;
    }

}
```

4) Right-click on ‘UDFExample.java’ and select <Run As-> ODPS UDF|UDTF|UDAF>:



5) Configure the following dialog box:


ODPS UDF|UDTF|UDAF Run Configuration

ODPS UDF|UDTF|UDAF Run Configuration

Class

odps.UserUDF

Select ODPS Project

odpstest_ay52c_ay52
example_project

Input Table

Table: wc_in1

Partitions:

Columns: col1,col2

e: p1=1,p2=1 (default all partitions)

e: c1,c2,c3 (default all columns)

6) Get the result:

```
ss2s:A1,A2
ss2s:A1,A2
ss2s:A1,A2
ss2s:A1,A2
```

Here just gives UDF running example. The running method of UDTF is the same as UDF, so it will not be introduced again.

Appendix

Escape Character

In ODPS SQL, a string constant can be quoted by single or double quotation marks. The string quoted by single quotation marks can contain double quotation marks or the string quoted by double quotation marks can contain single quotaton marks. Otherwise, you must use escape character to indicate it. The following expressions are acceptable:

```
"I'm a happy manong!"
'I\'m a happy manong!'
```

In ODPS SQL, '\' is a kind of escape character, used to express the special character in string or express its followed chracters 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, system will convert it to corresponding characters according to ASCII value. The following table lists some special characters:

Escape Character	Character
\b	backspace
\t	tab
\n	newline
\r	carriage-return
\’	Single quotation mark
\”	Double quotation marks
\ \	back-slant
\;	semicolon
\Z	control-Z
\0 or \00	tailed

```
select length('a\tb') from dual;
```

The result is 3, which indicates that there are three characters in the string. '\t' is considered as one character. Other characters followed behind it are expressed as themselves.

```
select 'a\ab',length('a\ab') from dual;
```

The result: 'aab', 3. '\a' is expressed as general 'a'.

Character Matching Description of LIKE

In LIKE matching, '%' indicates matching any multiple characters. ' ' (space) indicates matching a single character. To match '%' or ' ' itself, you must escape it. '\%' matches the character '%' and '\' matches the character ' ' (space).

```
'abcd' like 'ab%' -- true
'abcd' like 'ab\%' -- false
'ab%cd' like 'ab\\%%' -- true
```

📖 **Note:**

- About character set of string, ODPS SQL only supports the UTF-8 character set. If the data is encoded in other formats, maybe the calculation result is not correct.

Regular Expression

The regular expressions in ODPS SQL use PCRE standard, matched by character. The metacharacter to be supported are shown as follows:

Metacharacter	Description
^	Top of line (TOL)
$	End of line
.	Any character
*	Matches for zero or more times
+	Matches for once or more times
?	Matches for zero or once
?	Matches modifier. When this character follows any other constraints (*, +,? {n}, {n, {n, m},}, the match mode is non greedy. Non greedy mode matches strings as little as possible, while the default greedy mode matches strings as more as possible.
A|B	A or B
(abc)*	Matches 'abc' for zero or more 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.
[::]	Refer to 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 whitespace characters	[\x21-\x7E]
[[:lower:]]	Lowercase characters	[a-z]
[[:print:]]	[:graph:] and whitespace characters	[\x20-\x7E]
[[:punct:]]	punctuation	[][!"#$%&'()*+,./:;<=>?@\^_`{|}~-]
[[:space:]]	Whitespace characters	[\t\r\n\v\f]
[[:upper:]]	Uppercase characters	[A-Z]
[[:xdigit:]]	hexadecimal character	[A-Fa-f0-9]

Because system uses a backslash " \" as an escape character, all " \" which appear in the regular expression pattern will do two escapes. For example, the regular expression needs to match the string "a+b". "+" is a special character in regular expression and should be expressed by escape. The expression in regular engine is "a+b", because 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 existent:

```
select 'a+b' rlike 'a\\+b' from test_dual;


+------+
| _c1   |
+------+
| true |
+------+
```

In extreme case, to match the character “ \”, because “ \” is a special character in regular engine, it needs to be expressed by “ \”, while system will do an escape for it again, it will be written to be “ \\”.

```
select 'a\\b', 'a\\b' rlike 'a\\\\b' from test_dual;

+-----+------+
| _c0 | _c1   |
+-----+------+
| a\b | true |
+-----+------+
```

If TAB exists in a string, when system reads these two characters ‘ \t’, they are already saved as one character by system. So in regular expression, it is a general character.

```
select 'a\tb', 'a\tb' rlike 'a\tb' from test_dual;

+---------+------+
| _c0         | _c1   |
+---------+------+
| a         b | true |
+---------+------+
```

Reservrd Words

The following shows all reserved words in ODPS SQL. It colud not be used to nominate table, column or partition; otherwise an error will be thrown. Reserved words are not case sensitive.

```
%       &       &&       (       )       *       +
-       .       /        ;       <       <=      <>
=     >     >=     ?     ADD     AFTER     ALL
ALTER     ANALYZE     AND     ARCHIVE     ARRAY     AS     ASC
BEFORE     BETWEEN     BIGINT     BINARY     BLOB     BOOLEAN     BOTH
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
PARTITIONPROPERTIES    PARTITIONS
PERCENT    PLUS    PRECEDING    PRESERVE    PROCEDURE    PURGE
RANGE
RCFILE    READ    READONLY    READS    REBUILD    RECORDREADER
RECORDWRITER
REDUCE    REGEXP    RENAME    REPAIR    REPLACE    RESTRICT
REVOKE
RIGHT    RLIKE    ROW    ROWS    SCHEMA    SCHEMAS    SELECT
SEMI    SEQUENCEFILE    SERDE    SERDEPROPERTIES    SET    SHARED
SHOW
SHOW_DATABASE    SMALLINT    SORT    SORTED    SSL    STATISTICS
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
```

Revision History

Revision NO.	Revised Date	Revised by	Reason for Revision
V1.0	2015-4-2	Mao.jing	
V1.1	2015-5-5	Mao.jing	Add Chapter 5 ODPS Graph