ODPS 产品帮助文档目录

一、ODPS 介绍

（一）、ODPS 简介

1、什么是 ODPS

ODPS 提供海量数据处理及分析服务，让用户远离大数据运算烦恼。

开放数据处理服务（Open Data Processing Service，ODPS）是基于飞天内核构建的海量数据处理和分析的服务平台，它以 RESTful API 形式提供服务，具有 PB 级别的数据处理能力，主要适用于实时性要求不高的海量数据处理，如数据分析、海量数据统计、数据挖掘和商业智能领域。

ODPS 提供了数据上传下载通道，SQL 处理操作，并且提供了完善的安全解决方案，其包括的功能有:
数据通道：提供高并发的数据上传下载服务
SQL 计算
安全：给 ODPS 里所有的对象提供安全服务

2、概念解读

2.1 项目空间(Project)

用户空间 Project(有时也称项目)是 ODPS 的基本组织单元，它很类似传统数据库的 Database 或 Scheme 的概念，它是进行多用户隔离和访问控制的主要边界。在 ODPS 中，所有对象都是属于某个项目空间的。一个用户可以同时拥有多个项目空间的权限。

项目空间有多个属性，通过配置这些属性，可以达到控制项目空间及项目空间下所有对象行为的目的。目前 ODPS 对外开放的属性包括：

属性名称	描 述	默认值	取值范围
最大显示屏数	select 语句显示的最多记录行数，默认值 1000, 即 select 回结果小于 1000 行时，返回实际记录数， 大于 1000 行时，仅返回前 1000 行数据。	1000	1 ~ 5000
项目空间描述	用户自添加的项目空间备注。不会改变项目空间行为。		

用户可以通过”use <project_name>;”命令进入一个项目空间，例如：
use test_project; -- 进入一个名为 test_project 的项目空间

备注：此命令运行后，用户会进入一个名为”test_project”的项目空间，从而可以操作该项目空间下的对象，例如：表、实例等，而不需要关心操作对象所在的项目空间。”use

<project_name>”是属于 ODPS 命令。

2.2 表（Table）

表是 ODPS 的数据存储单元，类似于关系数据库中的表，它在逻辑上也是由行和列组成的二维结构，每行代表一条记录，每列表示相同数据类型的一个字段，一条记录可以包含一个或多个列，各个列的名称和类型构成这张表的 schema，比如一条记录包含以下字段
- user_id BIGINT，标识唯一用户 ID
- view_time BIGINT1，表示页面访问时间戳
- page_url STRING，页面 URL
- referrer_url STRING，来源 URL
- IP STRING，请求访问的机器 IP

在 ODPS 中，所有的数据都被存储在表中，表是由行和列组成的二维数据结构。每行代表一条记录，不同的列由列名和列数据类型标识，如下表所示：

SHOP_NAME	REVENUE
SHOP1	1000
SHOP2	1500

ODPS 中表的概念与 MYSQL，ORACLE 中表的概念基本等同。表中的任意列可以是 ODPS 支持的任意种数据类型(Bigint，Double，String，Boolean，Datetime)。在 ODPS 中的各种不同类型任务的操作（输入、输出）对象都是表。用户可以创建表，删除表以及向表中导入数据。为了提高处理效率，可以在创建表时进行分区(Partition)，即指定表内的某几个字段作为分区列。这时数据可以看成由逻辑上许多独立的块组成，在使用数据时如果指定了需要访问的分区名称，则只会读取相应的分区，避免全表扫描。

2.3 数据类型

ODPS 表中的列必须是下列描述的任意一种类型，各种类型的描述及取值范围包括：
各种数据类型均可以为 NULL
ODPS 不支持如 Array 这样复杂的数据类型。

类型	描述	取值范围
Bigint	8 字节有符号整型。请不要使用整型的最小值(-9223372036854775808)，这是系统保留值。	-9223372036854775807 ~ 9223372036854775807
String	字符串，支持 utf-8 编码。其他编码的字符行为未定义。 STRING 类型允许最长为 2M 字节。	
Boolean	布尔型。	True/False
Double	8 字节双精度浮点数。	-1.0 * 10^308 ~ 1.0 * 10^308
Datetime	日期类型。使用东八区时间作为系统标准时间。	0001-01-01 00:00:00 ~ 9999-12-31 23:59:59

2.4 实例(Instance)

在 ODPS 中，任意一条 SQL 与一个 ODPS 实例(下文简称实例或 Instance)一一对应。实例会经历运行(Running)及结束(Terminated)两个阶段。运行阶段的状态为 Running，而结束阶段的状态将会是 Success(成功)，Failed(失败)，Canceled(被取消)中的某一种。用户可以根据运行 SQL 时给出的 ODPS 实例 ID 查询、改变任务的状态，例如：

```
status <instance_id>;          --查看某 Instance 的状态
kill <instance_id>;            --停止某 Instance，将其状态设置为 Canceled
```

2.5 安全(Security)

在 ODPS 中访问任何数据，运行作业都必须有相应的权限。ODPS 提供 ACL 和 Policy 两种授权方式，详见相关说明。

2.6 分区（Partition）

指一张表下，根据分区字段（一个或多个组合）对数据存储进行划分。当利用分区字段对表进行分区时，新增分区、更新分区内数据和读取分区数据均不需要做全表扫描，可以提高处理效率。
针对海量数据，必须制定恰当的分区策略，通过选择常用的 where 列作为分区键。每个分区的容量没有限制，如果分区数量不宜过多，可能会导致后续操作不方便，一个 SQL 涉及的分区有数量限制。

2.7 资源（Resource）

资源(Resource)是 ODPS 特有的概念，用户可以上传本地自定义的 JAR 包或文件作为资源，也可以将 Project 下的某张表作为资源。

（二）、功能特性

1、主要功能
ODPS 提供了基于云计算的海量数据分析和处理。它提供了以下几大功能模块：
MapReduce（MR）编程模型，熟悉 Hadoop 的开发人员可以轻松转到 ODPS 平台，快速开发高效强大的数据处理应用。
SQL 计算和存储过程，采用类似传统 SQL 语法，传统数据库 SQL 程序员可以轻松入门，使用 ODPS SQL 对数据进行查询分析。ODPS 存储过程支持使用变量、判断语句和循环，适用于较复杂的 SQL 查询逻辑。
Tunnel 服务，Tunnel 服务适用于各异构数据源（如 mysql、oracle 和本地数据）以及不同的数据同步工具和 ODPS 的数据交互。
Xlib，提供一系列基于 ODPS 平台的数据挖掘算法，包含统计分析和机器学习算法，适用于对性能要求很高的 BI 数据挖掘。（公测阶段不开放）
图(Graph)编程模型，它是面向迭代的分布式图处理框架，支持类似 Google Pregel 的 Java

编程接口，用户可以基于 Graph 编程模型开发高效的机器学习算法或数据挖掘算法。（公测阶段不开放）
RODPS，为了利用 R 工具丰富的算法包和强大的数据可视化功能，ODPS 提供了 RODPS 包，在 R 中安装 RODPS 包后，可以把 R 建模结果自动以 SQL 形式发布到 ODPS 中运行，这样可以充分利用 ODPS 的计算能力。（公测阶段不开放）

（三）、ODPS 常用命令语法

1、Use Project

命令格式:
USE <project_name>;
行为：
进入指定项目空间。可以在不指定 Project 前缀的情况下直接操作该项目空间下的对象，包括：Table。
项目空间不存在或当前用户不在此项目空间中，异常返回；
示例：
odps@ $project_name>use test_project;　　--test_project 是用户有权限访问的一个 project
OK　　--console 返回 OK
备注：
上面给出的是在 Console 中运行此命令的示例。所有的 ODPS 命令格式大小写不敏感。
在命令运行后，用户可以直接访问该 Project 下的对象。例如，假设 test_project 项目空间下有表 test_src，用户运行：
```
select * from test_src;
```
ODPS 会自动搜索项目空间 test_project 下的表，如果存在此表，返回表中的数据。如果此表不存在，异常退出。如果用户在 test_project 下想要访问另一项目空间 test_project2 下的表 test_src，则需要指定项目空间名：
```
select * from test_project2.test_src;
```
此时返回 test_project2 项目空间下的数据结果，而不是 test_project 下的 test_src 表数据。 ODPS 没有提供创建及删除项目空间的命令。用户可以通过管理控制台达对各自的项目空间完成更多的配置及操作，详细介绍请参考“管理控制台”的相关介绍。

2、Create Table

命令格式：
```
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], ...)]
 [AS select_statement]
 [LIFECYCLE days]

 CREATE TABLE [IF NOT EXISTS] table_name
 LIKE existing_table_name
```

行为：
创建一张表
备注：
表名与列名均对大小写不敏感；
表名，列名中不能有特殊字符，只能用英文的 a-z, A-Z 及数字和下划线’_’，且以字母开头，名称的长度不超过 128 字节，否则报错；
注释内容是长度不超过 1024 字节的有效字符串，否则报错；
对于该命令更详细的介绍请参阅 *创建表(CREATE TABLE)*；
示例：

```
CREATE TABLE IF NOT EXISTS sale_detail(
 shop_name      STRING,
 customer_id    STRING,
 total_price    DOUBLE)
PARTITIONED BY (sale_date STRING,region STRING);     --如果没有同名表存在，创建一张分区表 sale_detail
```

3、Drop Table

命令格式：

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

行为：
删除一张表
如果不指定 IF EXISTS 选项而表不存在，则返回异常；若指定此选项，无论表是否存在，皆返回成功。
描述：
table_name：要删除的表名；
示例：

```
DROP TABLE sale_detail;                -- 若表存在，成功返回；
DROP TABLE IF EXISTS sale_detail;      -- 无论是否存在 sale_detail 表，均成功返回；
```

4、Describe Table

命令格式：

```
DESC <table_name>;
DESCRIBE <table_name>;
```

行为：
返回指定表的信息。具体返回包括：Owner(表的属主)，Project(表所属的项目空间)，CreateTime(创建时间)，LastDDLTime(最后一次 DDL 操作时间)，LastModifiedTime(表中的数据最后一次被改动的时间，即最后一次改变数据的 SQL 操作的时间)，InternalTable(表示被描述的对象是 Table，总是显示 YES)，Size(表数据所占存储容量的大小，单位 Bit)，Native Columns(非分区列的信息，包括：列名，类型，备注)，Partition Columns(分区列信息，包括：分区名，类型，备注)。
参数：
table_name：表名或视图名称
示例：

```
odps@ project_name>DESC sale_detail;                -- 描述一张分区表

+------------------------------------------------------------------------------+
|     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                |
|
+------------------------------------------------------------------------------+
```

备注：

上面给出的是在 Console 中运行此命令的示例；

如果是不带分区的表，将不会显示 Partition Columns 相关信息；

如果描述的是一个视图(View)，将不显示 InternalTable 选项，而是 VirtualView 选项，其值总是为 YES。与此类似地，Size 选项将会被 ViewText 选项替代，表示 View 的定义，例如：

select * from src。关于视图的介绍请参考 *创建视图(CREATE VIEW)*

5、Show Tables

命令格式:
SHOW TABLES;
行为:
列出当前项目空间下所有的表
示例:
odps@ project_name>show tables;

ALIYUN$odps_user@aliyun.com:table_name
.....
备注:
上面给出的是在 Console 中运行此命令的示例;
ALIYUN 是系统提示符，表示用户是阿里云用户;
odps_user@aliyun.com 是用户名，表示该表的创建者;
table_name 是表名;
此操作会返回项目空间下所有的表，因此可能会产生多个 ODPS 请求。如果其中的某个请求出错，会返回出错请求的 RequestID 并结束命令运行。用户可使用这个 RequestID 寻求技术支持。

6、Show Partitions

命令格式:
SHOW PARTITIONS <**table_name**>;
行为:
列出一张表的所有分区;
参数:
table_name:指定查询的表名称。表不存在或非分区表报错;
示例:
odps@ project_name>SHOW PARTITIONS table_name;
partition_col1=col1_value1/partition_col2=col2_value1
partition_col1=col1_value2/partition_col2=col2_value2
...
备注:
上面给出的是在 Console 中运行此命令的示例;
partition_col1 和 partition_col2 表示该表的分区列;
col1_value1，col2_value1，col1_value2，col2_value2 表示分区列对应的值;

7、Upload/Download Data

ODPS 提供了内置上传下载数据的命令，方便用户将本地数据文件导入到 ODPS 表中。使用方式:
配置操作：编辑 plugins/tunnel/plugin.ini 文件

endpoint=http://12.12.96.32 ：指定 Tunnel 服务的 endpoint
col.delimiter=44 ：指定列分隔符，默认为 44 ","
row.delimiter=10 ：指定列分隔符，默认为 10 "\n"
null.indicator=NULL ：指定空字符串，默认为空串。如果指定空值，可以把此行删除掉
date.format=yyyyMMddHHmmss ：指定日期的 format 格式，默认为 yyyyMMddHHmmss
bad.discard=false ：指定是否忽略脏数据，默认为 false，
命令说明：
此命令是 ODPS 为用户提供的对 ODPS Tunnel 服务的封装。请不要混淆 Update/Download Data 命令和 Tunnel 服务的概念。有关 Tunnel 服务的介绍，请参考 *ODPS TUNNEL 参考手册* 。Update/Download Data 命令的默认日期格式与 ODPS SQL 的默认日期格式有区别。ODPS SQL 的默认日期格式为:yyyy-mm-dd hh:mi:ss，详细信息请参阅 *String 类型与 Datetime 类型之间的转换* 。
当表中已经存在数据时，再次向表中导入数据，新导入的数据会添加到表中，原有的数据依然有效。
如果没有配置 null.indicator，表示空串”“为 NULL。
上传的本地数据文件最大为 300M。
bad.discard 指定是否忽略脏数据，默认值为 false，遇到脏数据时直接报错，提示脏数据所在行号及其他相关信息，脏数据所在行之前的数据不会被导入 ODPS 中。值为 true 时，忽略所有脏数据，脏数据对应的列会被赋值为 NULL。在数据导入完成时，提示脏数据行数。
脏数据的定义包括：多列，少列，某列的数据无法转换为对应列的格式，例如无法转换为 double，bigint，datetime 类型。
备注

用户可以通过修改 *快速开始* 中示例的数据文件格式，了解详细的错误提示。
上传下载命令格式：
UPLOAD <tablename> [PARTITION(partition_spec)] FROM <filepath>;

DOWNLOAD <tablename> [PARTITION(partition_spec)] TO <filepath>;
说明：
tablename: 数据上传或下载的表名；
PARTITION(partition_spec): 指定分区值。其格式如：partition(pt='123455', ds='23232323')。此时，将数据导入此分区中；
filepath: 本地文件路径，支持相对路径；

8、Show Instances

命令格式：
SHOW INSTANCES [**FROM** startdate **TO** enddate] [number]
行为：
返回由当前用户创建的 Instance 信息。
参数：
startdate, enddate：指定返回的 Instance 的时间段，即从起始时间 startdate 到结束时间 enddate 的 Instance 信息，需满足如下格式:yyyy-mm-dd hh:ms:ss。可选参数，若不指定，返回用户三天内提交的 Instance；

number:指定返回 Instance 数量。依照时间排序，返回离当前时间最近的 number 个 Instance 信息。若不指定 number，返回满足要求的所有 Instance 信息；
输出项包括:StartTime(时间精确到秒),RunTime(s),Status(Instance 状态，包括 Success, Failed, Running, Canceled)，InstanceID 以及 Instance 对应的 SQL：

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

备注：
上面给出的是在 Console 中运行此命令的示例；

9、Status Instance

命令格式：
STATUS <instance_id>;
行为：
返回指定 Instance 的状态，状态包括：Success, Failed, Running, Canceled;
如果此 Instance 并非当前用户创建，异常返回；
描述：
instance_id: Instance 的唯一标识符。指定查询哪个 Instance 状态。
示例：
odps@ $project_name>status 20131225123302267gk3u6k4y2;
Success
查看 ID 为 20131225123302267gk3u6k4y2 的 Instance 的状态，查询结果为 Success。

备注：
上面给出的是在 Console 中运行此命令的示例；

10、Kill Instance

命令格式:
kill <instance_id>;
行为：
停止用户指定的 Instance，此 Instance 的状态必须为 Running;
参数：
instance_id：Instance 的唯一标识符。必须是状态为 Running 的 Instance 的 ID，否则抛异常；
示例：
odps@ $project_name>kill 20131225123302267gk3u6k4y2;
OK
停止 ID 为 20131225123302267gk3u6k4y2 的 Instance。

备注：

上面给出的是在 Console 中运行此命令的示例；

11、Log

ODPS 的所有任务都是在分布式环境下运行的。通过 log 命令，用户可以查看任务中每个进程的状态，命令格式：

```
log <action> [OPTIONS]
```

其中 action 包括：get, help, ls, status, sum。每个 action 的功能可以通过 help action 查看，例如查看 ls action 的功能：

```
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>
```

下面给出使用 log 命令的示例：

假设存在表，运行如下 SQL：

```
SELECT CAST(sale_date AS DATETIME) FROM sale_detail;
   -- 这是一个会在运行时报错的 ODPS SQL，失败原因会在后续章节中涉及。
   -- 在这里，用户可以先不必关心失败的原因。
```

得到如下格式输出：

```
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'
```

查看这个 SQL 的每个进程的状态：

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

如果您对 SQL 的分布式实现方式不够了解，可以忽略第一项”M1_Stg1#0_0”展示的意义，但请注意这个字段能够唯一标识 SQL 中的进程。这一项表示了 ODPS SQL 的执行逻辑，不在本章的介绍范围内。第二列表示每个进程启动的时间。第三项表示每个进程工作的时间(由于示例中处理的数据量很少，只有两行数据，因此执行时间不足 1 秒)。最后一列”Failed”表示进程的状态为失败。

查看进程的 stdout 和 stderr：

```
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;
SetEnv - LD_LIBRARY_PATH:/apsara/lib64:  ......
SetEnv - PATH:/usr/ali/bin:/sbin:/usr/sbin:/bin:/usr/bin
SetEnv - PYTHONHOME:/usr/ali/python-2.7
SetEnv - PYTHONPATH:job_runtime
SetEnv - PYTHON_USER_PATH:.
LimitKey:core
SetLimit - core:unlimited
[2014-01-16 14:56:45.456484] ---------- Run Task: M1_Stg1#0 ----------
```

在这个示例中，从 stdout 和 stderr 处无法看到有帮助的出错信息。但在某些时候，stdout 和 stderr 的信息可以帮助用户调查问题。我们建议有分布式经验的用户使用此功能。

备注：

Log 的功能十分强大，在示例中我们我们仅仅展示了最基础的使用场景。请用户可以通过 help 命令获取更加详细的使用说明。

12、Whoami

显示用户信息，命令格式：

```
whoami;
```

显示的信息包括：云账号，endpoint，项目空间名称，示例：

```
odps@ test_project> whoami;
Name: ALIYUN$test_user@aliyun.com
End_Point: http://service.odps.aliyun.com/api
Project: test_project
```

二、ODPS CLT(command line tools) 参考手册

（一）、ODPS CLT 概要

1、文档说明

- 本文档是 ODPS 客户端(以下简称为 odpscmd)用户手册的一部分，说明如何借助 odpscmd 命令行工具使用 ODPS 服务的基础功能。
- 需要了解 odpscmd 的安全相关命令，请参阅 *ODPS 安全参考手册* ；需要了解 odpscmd 支持的 SQL 语法，请参阅 *ODPS SQL 参考手册* 。

Note

请不要依赖 odpscmd 的输出格式来做的解析工作。客户端的输出格式不承诺向前兼容。

2、下载 odpscmd（未开放）

3、安装 JRE

ODPS 客户端是一个 java 程序，需要 JRE 环境才能运行，请下载并安装 JRE 1.6 以上版本。

4、配置和运行 odpscmd

解压下载的 odps_clt_release_64.tar.gz 文件，可以看到如下 4 个文件夹：

```
bin/  conf/  lib/  plugins/
```

在 conf 文件夹中有 odps_conf.ini 文件。编辑此文件，填写相关信息：

```
project_name=${your_project_name}

    #指定您想进入的项目空间。此项可忽略,缺省时进入console后需要使用"use project"命令进入您的项目空间。

access_id=

access_key=

    #这两项是由云账号提供的，在阿里云官网上可以获取

end_point=http://service.odps.aliyun.com/api

    #链接 ODPS 使用的 URL
```

修改好配置文件后运行 bin 目录下的 odpscmd(在 LINUX 系统下是./bin/odpscmd，WINDOWS 下运行./bin/odpscmd.bat)，现在可以运行 ODPS 命令，如：

```
odps@ test_project> whoami;

Name: ALIYUN$test_user@aliyun.com

End_Point: http://service.odps.aliyun.com/api
```

```
Project: test_project
```

其中 odps 为标识符，test_project 为项目空间名称。

（二）、ODPS 客户端配置参数

1、查看帮助信息

用途：查看 odpscmd 的帮助信息。

使用方式：

1. 命令参数。可以通过使用- -help 或者-h 选项查看帮助信息，命令格式如：

```
odpscmd --help

odpscmd -h
```

2. 输入 odpscmd 命令”help;”或者”h;”。

示例：

```
$./bin/odpscmd -h

Usage: odpscmd [OPTION]...

where options include:

    -h, --help                              print this help message

    -v, --version                           print product version and exit



    --project=<prj_name>                    use project

    --endpoint=<http://host:port>           set endpoint

    -u <user_name> -p <password>            user name and password
```

```
    -f <"file_path;">                          execute command in file

    -e <"command;[command;]...">               execute command, include sql command


where -e option  include:

    -e "quit;"                                                  quit


    -e "list projects;"                                         list projects

    -e "use <prj_name>;"                                        open project


    -e "show tables;"                                           list tables in
current project

    -e "show partitions <table_name>;"                          list partitions

    -e "desc <table_name>;"                                     show table schema


    -e "read  <table_name> [<(col_name>[,..])][PARTITION
<(partition_spec)>][line_num];"

    SQL:

    -e "<sql>;"                                                 execute sql


    Instances:
```

```
    -e "show p [from startdate to enddate] [number];"          show instances, date format: eg. 2012-08-27

    -e "kill <instanceid>;"                                    stop instance

    -e "status <instanceid>;"                                  get instance status

    -e "log <action> [OPTIONS];"                               inspect instance logs
```

2、 Project 选项

用途：进入到用户指定的 project。

使用方式：

1. 在命令行中指定- -project 选项，如：

```
odpscmd --project=<project_name>
```

2. 在 odps_conf.ini 配置文件中添加 project_name 配置，如：

```
project_name=<project_name>
```

3、 Endpoint 选项

用途：连接指定的 ODPS 服务 URL。

使用方式：

1. 在命令行中指定- -endpoint 选项，如：

```
odpscmd  --endpoint=<end_point>
```

2. 在 odps_conf.ini 配置文件中添加 end_point 配置，如：

```
end_point=<end_point>
```

4、 ACCESS_ID 和 ACCESS_KEY 配置

用途：指定用户的 access_id 和 access_key。

使用方式：

1. 在命令行中指定-u 及-p 选项，如：

```
odpscmd -u <user_name> -p <password>
```

其中，user_name 即是 ACCESS_ID，password 即是 ACCESS_KEY。ACCESS_ID 和 ACCESS_KEY 可以从阿里云网站上获取。

2. 在 odps_conf.ini 配置文件中添加 access_id 和 access_key 配置，如：

```
access_id=<access_id>

access_key=<access_key>
```

5、指定命令执行脚本

用途：按顺序逐条执行脚本中的 odps 命令。

使用方式：通过-f 选项指定命令文件名，如：

```
odpscmd -f <file_name>
```

6、参数指定运行语句

用途：逐条执行-e 后面指定的 odps 命令。可以在-e 后加多条 ODPS 命令。

使用方式：

```
odpscmd -e <"odps_cmd">
```

7、查看 Version 选项

用途：查看 ODPS 客户端的版本信息。请注意，这个版本并不是 ODPS 服务的版本信息。

使用方式：

在命令行中指定- -version 或者-v 选项，如：

```
odpscmd --version

odpscmd -v
```

示例：

```
******Odps Command Line Tools******

BuildTime: 2013-12-25 15:06:46

Revision: 393787
```

三、ODPS SQL 手册

（一）、DDL 语法

1、创建表(CREATE TABLE)

语法格式

```
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
```

Note

- 表名与列名均对大小写不敏感。
- 在创建表时，如果不指定 IF NOT EXISTS 选项而存在同名表，则返回出错；若指定此选项,则无论是否存在同名表,即使原表结构与要创建的目标表结构不一致,均返回成功。已存在的同名表的元信息不会被改动。
- 数据类型只能是:bigint, double, boolean, datetime, string。
- 表名，列名中不能有特殊字符，只能用英文的 a-z,A-Z 及数字和下划线_，且以字母开头，名称的长度不超过 128 字节。
- PARTITIONED BY 指定表的分区字段，目前仅支持 String 类型。分区值不可以有双字节字符（如中文），必须是以英文字母 a-z,A-Z 开始后可跟字母数字，名称的长度不超过 128 字节。 如果名称中出现了\t, \n 等特殊字符会导致分区中的数据无法读出，允许的字符包括：空格 ‘ ‘，冒号’:’，下划线’_’，美元符’$’，井号’#’，点’.’，感叹号’!’和’@’。当利用分区字段对表进行分区时，新增分区、更新分区内数据和读取分区数据均不需要做全表扫描，可以提高处理效率。
- 注释内容是长度不超过 1024 字节的有效字符串。
- LIFECYCLE 指明此表的生命周期，关于生命周期的介绍请参考 *修改表的生命周期属性* 。CREATE TABLE LIKE 语句不会复制源表的生命周期属性。
- 目前，在表中建的分区层次不能超过 5 级。

在下面的例子中，创建表 sale_detail 保存销售记录，该表使用销售时间(sale_date)和销售区域(region)作为分区列：

```
CREATE TABLE IF NOT EXISTS sale_detail(

 shop_name     STRING,

 customer_id   STRING,

 total_price   DOUBLE)

PARTITIONED BY (sale_date STRING,region STRING);

    -- 创建一张分区表 sale_detail
```

也可以通过 CREATE TABLE … AS SELECT ..语句创建表，并在建表的同时将数据复制到新表中，如

```
CREATE TABLE sale_detail_ctas1 AS

   SELECT * FROM sale_detail;
```

此时，如果 sale_detail 中存在数据，上面的示例会将 sale_detail 的数据全部复制到 sale_detail_ctas1 表中。但请注意，此处 sale_detail 是一张分区表，而通过 CREATE TABLE ... AS SELECT ... 语句创建的表不会复制分区属性，只会把源表的分区列作为目标表的一般列处理，即 sale_detail_ctas1 是一个含有 5 列的非分区表。

在 CREATE TABLE ... AS SELECT ...语句中，如果在 SELECT 子句中使用常量作为列的值，建议指定列的名字，例如：

```
CREATE TABLE sale_detail_ctas2 AS

   SELECT shop_name,

       customer_id,

       total_price,

       '2013' AS sale_date,

       'China' AS region

   FROM sale_detail;
```

如果不加列的别名，如：

```
CREATE TABLE sale_detail_ctas3 AS

   SELECT shop_name,

       customer_id,

       total_price,
```

```
        '2013',
        'China'
    FROM sale_detail;
```

则创建的表 sale_detail_ctas3 的第四、五列会是类似”_c3”，“_c4”这样的系统自动生成的名字，再次使用表 sale_detail_ctas3 时需要加上反引号才能正确引用，如：

```
SELECT `_c3`, `_c4` FROM sale_detail_ctas3;
```

直接执行 SQL “SELECT _c3, _c4 FROM test_tbl”会报错退出。因为 ODPS SQL 的列名不能够以”_”开头，因此必须加反引号”`”做特别处理。

如果希望源表和目标表具有相同的表结构，可以尝试使用 CREATE TABLE ... LIKE 操作，如：

```
CREATE TABLE sale_detail_like LIKE sale_detail;
```

此时，sale_detail_like 的表结构与 sale_detail 完全相同。除生命周期属性外，列名、列注释以及表注释等均相同。但 sale_detail 中的数据不会被复制到 sale_detail_like 表中。

2、删除表(DROP TABLE)

语法格式

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

Note

- 如果不指定 IF EXISTS 选项而表不存在，则返回异常；若指定此选项，无论表是否存在，皆返回成功。
- 在 console 的交互模式下运行 DROP TABLE 命令时，会有提示是否确认删除。

示例：

```
CREATE TABLE sale_detail_drop LIKE sale_detail;
```

```
DROP TABLE sale_detail_drop;
    --若表存在，成功返回；若不存在，异常返回；

DROP TABLE IF EXISTS sale_detail_drop2;
    --无论是否存在 sale_detail_drop2 表，均成功返回；
```

3、重命名表(RENAME TABLE)

语法格式

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

Note

- RENAME 操作仅修改表的名字，不改动表中的数据。
- 如果已存在与 new_table_name 同名表，报错。
- 如果 table_name 不存在，报错。

```
CREATE TABLE sale_detail_rename1 LIKE sale_detail;

ALTER TABLE sale_detail_rename1 RENAME TO sale_detail_rename2;
```

4、创建视图(CREATE VIEW)

语法格式

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

```
 [AS select_statement]
```

Note

- 创建视图时，必须有对视图所引用表的读权限。
- 视图只能包含一个有效的 SELECT 语句。
- 视图可以引用其它视图，但不能引用自己，也不能循环引用。
- 不可以向视图写入数据，例如使用 INSERT INTO 或者 INSERT OVERWRITE 操作视图。
- 当视图建好以后，如果视图的引用表发生了变更，有可能导致视图无法访问，例如：删除被引用表。用户需要自己维护引用表及视图之间的对应关系。
- 如果没有指定 IF NOT EXISTS，在视图已经存在时用 CREATE VIEW 会导致异常。这种情况可以用 CREATE OR REPLACE VIEW 来重建视图，重建后视图本身的权限保持不变。

示例：

```
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;
```

5、删除视图(DROP VIEW)

语法格式

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

Note

- 在 console 中的交互模式下运行 DROP VIEW 命令时，会有提示是否确认删除。
- 如果视图不存在且没有指定 IF EXISTS，报错。

示例：

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

6、重命名视图(RENAME VIEW)

语法格式

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

Note

- 如果已存在同名 View，报错。

示例：

```
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;
```

7、添加分区(ADD PARTITIONS)

语法格式

```
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec



partition_spec:

 : (partition_col1 = partition_col_value1, partition_col2 =
partiton_col_value2, ...)
```

Note

- 如果未指定 IF NOT EXISTS 而同名的分区已存在，则出错返回。
- 目前 ODPS 支持的分区数量上限为 20 万。
- 对于多级分区的表，如果想添加新的分区，必须指明全部的分区值。

示例：为 sale_detail 表添加一个新的分区，

```
ALTER TABLE sale_detail ADD IF NOT EXISTS PARTITION (sale_date='201312',
region='hangzhou');

  -- 成功添加分区，用来存储 2013 年 12 月杭州地区的销售记录。

ALTER TABLE sale_detail ADD IF NOT EXISTS PARTITION (sale_date='201312',
region='shanghai');

  -- 成功添加分区，用来存储 2013 年 12 月上海地区的销售记录。

ALTER TABLE sale_detail ADD IF NOT EXISTS PARTITION(sale_date='20111011');

  -- 仅指定一个分区 SALE_DATE，出错返回

ALTER TABLE SALE_DETAIL ADD IF NOT EXISTS PARTITION(region='shanghai');

  -- 仅指定一个分区 region，出错返回
```

8、删除分区(DROP PARTITION)

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

partition_spec:

: (partition_col1 = partition_col_value1, partition_col2 =
partiton_col_value2, ...)
```

Note

- 如果分区不存在且未指定 IF EXISTS，则出错返回。

示例：从表 sale_detail 中删除一个分区，

```
ALTER TABLE sale_detail DROP PARTITION(sale_date='201312',region='hangzhou');

 -- 成功删除 2013 年 12 月杭州分区的销售。
```

9、修改表的注释

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

Note

- table_name 必须是已存在的表；comment 最长 1024 字节；

示例：

```
ALTER TABLE sale_detail SET COMMENT

  'new coments for table sale_detail';
```

通过 ODPS 命令 DESC 可以查看表中 comment 的修改，请参阅 *Describe Table* 。

10、添加列

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

Note

- 列的数据类型只能是: bigint, double, boolean, datetime, string。

11、修改列名

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

Note

- old_col_name 必须是已存在的列；
- 表中不能有名为 new_col_name 的列；

12、修改列、分区注释

```
ALTER TABLE table_name CHANGE COLUMN col_name COMMENT 'comment';
```

Note

- comment 内容最长 1024 字节；

13、修改表的生命周期属性

ODPS 提供数据生命周期管理功能，方便用户释放存储空间，简化回收数据的流程。

语法格式：

```
ALTER TABLE table_name SET LIFECYCLE days;
```

Note

- days 参数为生命周期时间，只接受正整数。单位：天；

如果表 table_name 是非分区表，自最后一次数据被修改开始计算，经过 days 天后数据仍未被改动，则此表无需用户干预，将会被 ODPS 自动回收（类似 DROP TABLE 操作）。在 ODPS 中，每当表的数据被修改后，表的 LastModifiedTime 将会被更新，因此，ODPS 会根据每张表的 LastModifiedTime 以及 LIFECYCLE 的设置来判断是否要回收此表。如果 table_name 是分区表，则根据各分区的 LastModifiedTime 判断该分区是否该被回收。关于 LastModifiedTime 的介绍请参考 *Describe Table* 。

不同于非分区表，分区表的最后一个分区被回收后，该表不会被删除。生命周期只能设定到表级别，不能再分区级设置生命周期。创建表时即可指定生命周期，详情请参阅*创建表 (CREATE TABLE)* 。

示例：

```
CREATE TABLE test_lifecycle(key STRING) LIFECYCLE 100;

    -- 新建 test_lifecycle 表，生命周期为 100 天。

ALTER TABLE test_lifecycle SET LIFECYCLE 50;

    -- 修改 test_lifecycle 表，将生命周期设为 50 天。
```

14、修改表、分区的修改时间

ODPS SQL 提供 TOUCH 操作用来修改表或分区的”LastModifiedTime”。效果会将表或分区的”LastModifiedTime”修改为当前时间。

语法格式：

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

Note

- table_name 或 partition_col 不存在，则报错返回；
- 指定的 partition_col_value 不存在，则报错返回；
- 此操作会改变表的”LastModifiedTime”的值，此时，ODPS 会认为表或分区的数据有变动，生命周期的计算会重新开始。

（二）DML 语法

1、更新表中的数据(INSERT OVERWRITE/INTO)

语法格式：

```
INSERT OVERWRITE|INTO TABLE tablename [PARTITION (partcol1=val1,
partcol2=val2 ...)]

select_statement

FROM from_statement;
```

在 ODPS SQL 处理数据的过程中，INSERT OVERWRITE/INTO 用于将计算的结果保存目标表中。INSERT INTO 与 INSERT OVERWRITE 的区别是，INSERT INTO 会向表或表的分区中追加数据，而 INSERT OVERWRITE 则会在向表或分区中插入数据前清空表中的原有数据。在使用 ODPS 处理数据的过程中，INSERT OVERWRITE/INTO 是最常用到的语句，它们会将计算的结果保存一个表中，可以供下一步计算使用。如，可以用如下操作计算 sale_detail 表中不同地区的销售额

```
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;
```

需要注意的是，在进行 INSERT 更新数据操作时，源表与目标表的对应关系依赖于在 SELECT 子句中列的顺序，而不是表与表之间列名的对应关系，下面的 SQL 语句仍然是合法的：

```
INSERT OVERWRITE TABLE sale_detail_insert PARTITION (sale_date='2013',
region='china')

   SELECT customer_id, shop_name, total_price FROM sale_detail;

    -- 在创建 sale_detail_insert 表时，列的顺序为：

    -- shop_name string, customer_id string, total_price bigint

    -- 而从 sale_detail 向 sale_detail_insert 插入数据是，sale_detail 的插入顺序为：

    -- customer_id, shop_name, total_price

    -- 此时，会将 sale_detail.customer_id 的数据插入 sale_detail_insert.shop_name

    -- 将 sale_detail.shop_name 的数据插入 sale_detail_insert.customer_id
```

2、多路输出(MULTI INSERT)

ODPS SQL 支持在一个语句中插入不同的结果表或者分区

语法格式：

```
FROM from_statement

INSERT OVERWRITE | INTO TABLE tablename1 [PARTITION (partcol1=val1,
partcol2=val2 ...)]

  select_statement1

[INSERT OVERWRITE | INTO TABLE tablename2 [PARTITION ...]

  select_statement2]
```

Note

- 一般情况下，单个 SQL 里最多可以写 128 路输出，超过 128 路报语法错误。

- 在一个 MULTI INSERT 中，对于分区表，同一个目标分区不可以出现多次；对于未分区表，该表不能出现多次。
- 对于同一张分区表的不同分区，不能同时有 INSERT OVERWRITE 和 INSERT INTO 操作，否则报错返回。

如，

```
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;

   -- 成功返回,将 sale_detail 的数据插入到 sales 里的 2010 年及 2011 年中国大区的销售记录中

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;

   -- 出错返回，同一分区出现多次

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;

   -- 出错返回，同一张表的不同分区，不能同时有 INSERT OVERWRITE 和 insert into 操作
```

3、输出到动态分区(DYNAMIC PARTITION)

在 INSERT OVERWRITE 到一张分区表时，可以在语句中指定分区的值。也可以用另外一种更加灵活的方式，在分区中指定一个分区列名，但不给出值。相应的，在 SELECT 子句中的对应列来提供分区的值。

语法格式：

```
INSERT OVERWRITE TABLE tablename PARTITION (partcol1, partcol2 ...)

select_statement FROM from_statement;
```

Note

- 目前，在使用动态分区功能的 SQL 中，在分布式环境下，单个进程最多只能输出 512 个动态分区，否则引发运行时异常；
- 在现阶段，任意动态分区 SQL 不可以生成超过 2000 个动态分区，否则引发运行时异常；
- 动态生成的分区值不可以为 NULL，否则会引发异常；
- 如果目标表有多级分区，在运行 INSERT 语句时允许指定部分分区为静态，但是静态分区必须是高级分区；

下面，我们使用一个简单的例子来说明动态分区：

```
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;
```

按照这种写法，在 SQL 运行之前，是不知道会产生哪些分区的，只有在 SELECT 运行结束后，才能由 region 字段产生的值确定会产生哪些分区，这也是为什么叫做”动态分区”的原因。

其他示例：

```
CREATE TABLE sale_detail_dypart LIKE sale_detail;

INSERT OVERWRITE TABLE sale_detail_dypart PARTITION (sale_date, region)

    SELECT * FROM sale_detail;

    -- 成功返回

INSERT OVERWRITE TABLE sale_detail_dypart PARTITION (sale_date='2013', region)

    SELECT shop_name,customer_id,total_price,region FROM sale_detail;

    -- 成功返回，多级分区，指定一级分区

INSERT OVERWRITE TABLE sales PARTITION (region='china', sale_date)

    SELECT shop_name,customer_id,total_price,region FROM sale_detail;

    -- 失败返回，不能仅指定低级子分区，而动态插入高级分区
```

4、SELECT 操作

语法格式：

```
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]
```

在使用 SELECT 语句时需要注意如下几点：

1、SELECT 操作从表中读取数据，要读的列可以用列名指定，或者用*代表所有的列，一个简单的 SELECT 如下。ODPS 可以直接将 SELECT 的结果显示在 console 中:

```
SELECT * FROM sale_detail;
```

或者只读取 sale_detail 的一列 shop_name

```
SELECT shop_name FROM sale_detail;
```

在 WHERE 中可以指定过滤的条件，如

```
SELECT * FROM sale_detail

WHERE shop_name LIKE 'hang%';
```

2、在 SELECT 语句的 WHERE 条件中可以指定分区范围，这样可以仅仅扫描表的指定部分，避免全表扫描。如下所示，

```
SELECT sale_detail.*

FROM sale_detail

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

3、在 table_reference 中支持使用嵌套子查询，如：

```
SELECT * FROM (SELECT region FROM sale_detail) t WHERE region = 'shanghai';
```

4、如果有重复数据行时，在字段前使用 DISTINCT，会将重复字段去重，只返回一个值，而使用 ALL 将返回字段中所有重复的值，不指定此选项时默认效果和 ALL 相同。使用 DISTINCT 只返回一行记录，如

```
SELECT DISTINCT region FROM sale_detail;
```

5、GROUP BY：在 SELECT 中包含聚类函数时，用 GROUP BY 指定分类的列；必须写列的完整表达式，不可用列的 column-alias，如

```
SELECT region FROM sale_detail GROUP BY region;
```

Note

- 关于聚合函数的介绍请参考 *聚合函数*

6、在使用 ORDER BY 排序时，NULL 会被认为比任何值都小，这个行为与 MYSQL 一致，但是与 ORACLE 不一致。ORDER BY 后面必须加 LIMIT。ORDER BY 后面必须加列的别名，当 SELECT 某列时，如果没有指定列的别名，将列名作为列的别名。

7、[LIMIT number]的 number 是常数，限制输出行数。当使用无 LIMIT 的 SELECT 语句直接从屏幕输出查看结果时，最多只输出 1000 行。每个项目空间的这个屏显最大限制限制可能不同，可以通过控制台面板控制。

8、SORT BY 前必须加 DISTRIBUTE BY;

9、 ORDER BY 不和 DISTRIBUTE BY/SORT BY 共用，同时 GROUP BY 也不和 DISTRIBUTE BY/SORT BY 共用;

5、子查询

普通的 SELECT 是从几张表中读数据，如 SELECT column_1, column_2 ... FROM table_name，但查询的对象也可以是另外一个 SELECT 操作，如:

```
SELECT * FROM (SELECT shop_name FROM sale_detail) a;
```

Note

- 子查询必须要有别名。

在 FROM 子句中，子查询可以当作一张表来使用，与其它的表或子查询进行 JOIN 操作，如

```
CREATE TABLE shop AS SELECT * FROM sale_detail;

SELECT a.shop_name, a.customer_id, a.total_price FROM

 (SELECT * FROM shop) a;
```

6、UNION ALL

语法格式：

```
select_statement UNION ALL select_statement
```

将两个或多个 SELECT 操作返回的数据集联合成一个数据集，如果结果有重复行时，会返回所有符合条件的行，不进行重复行的去重处理。需要注意的是：ODPS SQL 不支持顶级的两个查询结果合并，要改写为一个子查询的形式，如

```
SELECT * FROM sale_detail WHERE region = 'hangzhou'

  UNION ALL

SELECT * FROM sale_detail WHERE region = 'shanghai';
```

需要改成:

```
SELECT * FROM (

  SELECT * FROM sale_detail WHERE region = 'hangzhou'

    UNION ALL

  SELECT * FROM sale_detail WHERE region = 'shanghai')

tmp;
```

Note

- UNION ALL 操作对应的各个子查询的列个数、名称和类型必须一致。如果列名不一致时，可以使用列的别名加以解决。
- 一般情况下，ODPS 最多允许 128 个表的 UNION ALL，超过此限制报语法错误。

7、JOIN 操作

```
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 是一个等式表达式

LEFT JOIN 会从左表(shop)那里返回所有的记录，即使在右表(sale_detail)中没有匹配的行。

```
SELECT * FROM shop a LEFT OUTER JOIN sale_detail b ON a.shop_name=b.shop_name;
```

RIGHT OUTER JOIN 右连接，返回右表中的所有记录，即使在左表中没有记录与它匹配，例如:

```
SELECT * FROM shop a RIGHT OUTER JOIN sale_detail b ON a.shop_name=b.shop_name;
```

FULL OUTER JOIN 全连接, 返回左右表中的所有记录，例如:

```
SELECT * FROM shop a FULL OUTER JOIN sale_detail b ON a.shop_name=b.shop_name;
```

在表中存在至少一个匹配时，INNER JOIN 返回行。 关键字 INNER 可省略。

```
SELECT * FROM shop a INNER JOIN sale_detail b ON a.shop_name=b.shop_name;

SELECT * FROM shop a JOIN sale_detail b ON a.shop_name=b.shop_name;
```

连接条件，只允许 AND 连接的等值条件。只有在 MAPJOIN 中，可以使用不等值连接或者使用 OR 连接多个条件。并且最多支持 16 路 JOIN 操作。

8、MAPJOIN HINT

当一个大表和一个或多个小表做 JOIN 时,可以使用 MAPJOIN,性能比普通的 JOIN 要快很多。MAPJOIN 的基本原理是：在小数据量情况下，SQL 会将用户指定的小表全部加载到执行 JOIN 操作的程序的内存中，从而加快 JOIN 的执行速度。需要注意，使用 MAPJOIN 时，

- LEFT OUTER JOIN 的左表必须是大表
- RIGHT OUTER JOIN 的右表必须是大表
- INNER JOIN 左表或右表均可以作为大表
- FULL OUTER JOIN 不能使用 MAPJOIN
- MAPJOIN 支持小表为子查询;
- 使用 MAPJOIN 时需要引用小表或是子查询时，需要引用别名；
- 在 MAPJOIN 中，可以使用不等值连接或者使用 OR 连接多个条件；
- 目前 ODPS 在 MAPJOIN 中最多支持指定 6 张小表，否则报语法错误；
- 如果使用 MAPJOIN，则所有小表占用的内存总和不得超过 512M。

下面是一个简单的示例：

```
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 不支持支持在普通 JOIN 的 ON 条件中使用不等值表达式、OR 逻辑等复杂的 JOIN 条件，但是在 MAPJOIN 中可以进行如上操作，例如：

```
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;
```

9、CASE WHEN 表达式

ODPS 提供两种 CASE WHEN 语法格式，如下所述：

```
CASE value

 WHEN (_condition1) THEN result1

 WHEN (_condition2) THEN result2

 ...
```

```
 ELSE resultn

 END

CASE

  WHEN (_condition1) THEN result1

  WHEN (_condition2) THEN result2

  WHEN (_condition3) THEN result3

  ...

  ELSE    resultn

END
```

CASE WHEN 表达式可以根据表达式 value 的计算结果灵活返回不同的值， 如以下语句根据 SHOP_NAME 的不同情况得出所属区域

```
SELECT

CASE

WHEN shop_name IS NULL THEN 'default_region'

WHEN shop_name LIKE 'hang%' THEN 'zj_region'

END AS region

FROM sale_detail;
```

Note

- 如果 result 类型只有 bigint,double，统一转 double 再返回;
- 如果 result 类型中有 string 类型，统一转 string 再返回，如果不能转则报错（如 boolean 型）;
- 除此之外不允许其它类型之间的转换;

四、安全参考手册

（一）概述

1、目标用户群

主要面向 ODPS 项目空间 Owner、管理员以及对 ODPS 多租户数据安全体系感兴趣的用户。ODPS 多租户数据安全体系主要包括如下内容：用户认证、项目空间的用户与授权管理、跨项目空间的资源分享以及项目空间的数据保护。

2、快速入门

2.1 简介

如果你对 ODPS 安全体系了解不多，而希望能快速、正确使用 ODPS 安全管理功能，那么可以参照本节所列出的一些典型场景。

2.2 典型场景

- 添加一个新的项目组成员，并对他授权
- 添加多个具有相同角色的项目组成员，并授权他们访问存在的某张表
- 添加多个具有相同角色的项目组成员，并授权他们访问所有的表（包括未来可能创建的）
- 将资源打包分享给另一个项目空间
- 项目空间的数据保护设置

2.2.1 添加一个新的项目组成员，并对他授权

场景描述：Jack 是项目空间 prj1 的管理员，一个新加入的项目组成员 Alice（已拥有云账号： alice@gmail.com）申请加入项目空间 prj1，并申请如下权限：查看 table 列表，提交作业，创建 table。

操作步骤：（由项目空间管理员来操作）

```
use prj1;

add user ALIYUN$alice@gmail.com; --添加用户
```

```
grant List, CreateTable, CreateInstance on project prj1 to user
ALIYUN$alice@gmail.com; --使用 grant 语句对用户授权
```

Note

- 如果项目空间不支持 ALIYUN 账号，那么需要执行命令 add accountprovider ALIYUN; 以支持 ALIYUN 账号。

2.2.2 添加多个具有相同角色的项目组成员，并授权他们访问存在的某张表

场景描述：Jack 是项目空间 prj1 的管理员，有三个新加入的项目组成员：Alice, Bob, Charlie，他们的角色是数据审查员。他们要申请如下权限：查看 table 列表，提交作业，读取表 userprofile。

对于这个场景的授权，项目空间管理员可以使用 ACL 授权机制来完成。

操作方法：

```
use prj1;

add user ALIYUN$alice@gmail.com; --添加用户

add user ALIYUN$bob@gmail.com;

add user ALIYUN$charlie@gmail.com;

create role tableviewer; --创建角色

grant List, CreateInstance on project prj1 to role tableviewer; --对角色赋权

grant Describe, Select on table userprofile to role tableviewer; --对角色赋权

grant tableviewer to ALIYUN$alice@gmail.com; --对用户赋予角色 tableviewer

grant tableviewer to ALIYUN$bob@gmail.com;

grant tableviewer to ALIYUN$charlie@gmail.com;
```

2.2.3 添加多个具有相同角色的项目组成员，并授权他们访问所有的表（包括未来可能创建的）

场景描述：Jack 是项目空间 prj1 的管理员，有三个新加入的项目组成员：Alice, Bob, Charlie，他们的角色是数据审查员。他们要申请如下权限：查看 table 列表，提交作业，读取所有 table（包括未来可能创建的）。

对于这个场景的授权，项目空间管理员需要使用 Policy 授权机制，因为 Policy 授权机制可以使用通配符来描述授权对象。

操作方法：

Step 1: 创建一个 Policy 文件 /tmp/policy1.txt，内容如下：

```
{
"Version": "1",
"Statement":
 [{
    "Effect":"Allow",
    "Action":["odps:CreateInstance","odps:List"],
    "Resource":"acs:odps:*:projects/prj1"
}
,{
    "Effect":"Allow",
    "Action":["odps:Describe","odps:Select"],
    "Resource":"acs:odps:*:projects/prj1/tables/*"
}]
```

```
}
```

Note

- Policy 语法请参考”访问策略语言章节”。

Step 2: 添加用户，使用角色对用户授权

```
use prj1;

add user ALIYUN$alice@gmail.com; --添加用户

add user ALIYUN$bob@gmail.com;

add user ALIYUN$charlie@gmail.com;

create role tableviewer; --创建角色

put policy /tmp/policy1.txt on role tableviewer; --使用 policy 对角色赋权

grant tableviewer to ALIYUN$alice@gmail.com; --对用户赋予角色 tableviewer

grant tableviewer to ALIYUN$bob@gmail.com;

grant tableviewer to ALIYUN$charlie@gmail.com;
```

2.2.4 将资源打包分享给另一个项目空间

场景描述：Jack 是项目空间 prj1 的管理员。John 是项目空间 prj2 的管理员。由于业务需要，Jack 希望将其项目空间 prj1 中的某些资源（如 datamining.jar, 所有以”sample”开头的表）分享给 John 的项目空间 prj2。如果项目空间 prj2 的用户 Bob 需要访问这些资源，那么 prj2 管理员可以通过 ACL 或 Policy 自主授权，无需再麻烦 Jack。

操作方法：

Step 1: 在项目空间 prj1 中创建 package（由 prj1 owner 操作）

```
use prj1;
```

```
create package datamining; --创建一个 package

add resource datamining.jar to package datamining; --添加资源到 package

add table sample* to package datamining; --添加 table 到 package

allow project prj2 to install package datamining; --将 package 分享给项目空间 prj2
```

Step 2: 在项目空间 prj2 中安装 package（由 prj2 owner 操作）

```
use prj2;

install package prj1.datamining; --安装一个 package

describe package prj1.datamining; --查看 package 中的资源列表
```

Step 3: 项目空间 prj2 管理员对 package 进行自主授权（由 prj2 owner 或管理员操作）

```
use prj2;

grant Read on package prj1.datamining to user ALIYUN$bob@gmail.com; --通过 ACL 授权 Bob 使用 package

put policy /tmp/policy.txt; --通过 policy 授权所有用户使用 package
```

/tmp/policy.txt 中需要包含如下的一条 Statement：

```
{

   "Effect":"Allow",

   "Principal":"*",

   "Action":"odps:Read",

   "Resource":"acs:odps:*:projects/prj2/package/prj1.datamining"
```

```
}
```

2.2.5 项目空间的数据保护设置

场景描述：Jack 是项目空间 prj1 的管理员。该项目空间有很多敏感数据，比如用户身份号码和购物记录。而且还有很多具有自主知识产权的数据挖掘算法。Jack 希望能将项目空间中的这些敏感数据和算法保护好，项目中用户只能在项目空间中访问，数据只能在项目空间内流动，不允许流出到项目空间之外。

操作方法：（由项目空间 owner 操作）

```
use prj1;

set ProjectProtection=true; --开启项目空间的数据保护机制
```

一旦当项目空间开启数据保护机制后，使用 SQL, MR, DT, XLIB 都无法将项目空间中的数据转移到项目空间之外，所有的数据只能在项目空间内部流动。

但是在某些情况下，Alice 并不满足于站在受保护的项目空间内来访问数据表，可能由于业务需要，Alice 需要将某些数据表导出到项目空间之外，并且也经过空间管理员的审核通过。针对这类情况，ODPS 提供了两种机制来支持受保护项目空间的数据流出。

方法 1: 设置 ExceptionPolicy

Step 1: 创建一个 Policy 文件 /tmp/exception_policy.txt，只允许 Alice 使用 SQL 语句将 t1 从项目空间 prj1 导出。policy 内容如下：

```
{

"Version": "1",

"Statement":

 [{

   "Effect":"Allow",

   "Principal":"ALIYUN$alice@gmail.com",
```

```
    "Action":["odps:Describe","odps:Select"],
    "Resource":"acs:odps:*:projects/prj1/tables/t1",
    "Condition":{
        "StringEquals": {
            "odps:TaskType":"SQL"
        }
    }
 }]
}
```

Step 2: 设置 exception policy

```
use prj1;
--开启项目空间的数据保护机制，并设置数据导出的例外
set ProjectProtection=true with exception /tmp/exception_policy.txt;
```

方法 2: 设置 TrustedProject。将 prj2 设置为 prj1 的可信项目空间，设置后将允许 prj1 中的所有数据流出到 prj2。

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

Note

- 基于 Package 的资源打包分享机制和项目空间的数据保护机制是正交的两种安全机制。ODPS 规定：资源分享拥有更高的优先级。这就是说，在一个受保护的项目空间中，如果一个对象是通过 Package 机制分享给其它项目空间，那么跨项目空间访问该对象时将不受 ProjectProtection 规则的限制。

（二）ODPS 用户认证

1、简介

用户认证(Authentication)的主要功能是检查请求(request)发送者的真实身份。它一般包括两个方面：
1、正确验证消息发送方的真实身份；
2、正确验证接收到的消息在途中是否被篡改。

2、关于云账号认证

云账号认证使用消息签名机制，它不同于传统的基于用户名/密码的认证方法。消息签名机制可以保证消息在 HTTP 传输过程中的完整性(Integrity)和真实性(Authenticity)。常用的消息签名算法有 HMAC-SHA1 和 RSA-SHA1。传统的基于用户名/密码的认证方法适合于人机交互模式，如浏览 Web 网站；而消息签名机制则适合于非交互模式，比如编写 APP 应用程序访问开放服务的 API。一个云账号可以绑定一个或多个 AccessKey。如果一个云账号用户开发了多款 APP 应用程序，通常建议每一款 APP 应用程序使用不同的 AccessKey 并周期性更换。

3、云账号的使用方法

3.1 申请云账号

如果你还没有云账号，请访问 http://account.aliyun.com 以申请一个属于你的云账号。申请云账号时需要一个有效的电子邮箱地址，而且此邮箱地址将被当作云账号。比如，Alice 可以使用她的 alice@gmail.com 邮箱来注册一个云账号，那么她的云账号就是 alice@gmail.com。

3.2 申请 AccessKey

拥有云账号之后，可以登录访问 http://i.aliyun.com/access_key 以创建或管理当前云账号的 AccessKey 列表。一个 AccessKey 由两部分组成：AccessKeyId 和 AccessKeySecret。AccessKeyId 用于检索 AccessKey，而 AccessKeySecret 用于计算消息签名，所以需要严格保护以防泄露。当一个 AccessKey 需要更新时，用户可以创建一个新的 AccessKey，然后禁用老的 AccessKey。

3.3 使用云账号登录 ODPS

当使用 odpscmd 登录时，需要在配置文件 conf/odps_config.ini 中配置 AccessKey 相关信息，比如：

```
project_name=myproject

access_id=<这里输入 Access ID，不带尖括号>

access_key=<这里输入 Access Key，不带尖括号>

end_point=http://service.odps.aliyun-inc.com/api

project_name=myproject

access_id=<这里输入 Access ID，不带尖括号>

access_key=<这里输入 Access Key，不带尖括号>

end_point=http://service.odps.aliyun.com/api
```

3.4 相关问题

如何在你的项目空间中添加一个云账号、域账号或淘宝账号用户，请参考：

项目空间的用户与授权管理【基础篇：自主访问控制】

（三）项目空间的用户与授权管理【基础篇：自主访问控制】

1、简介

ODPS 自主访问控制机制(Discretionary Access Control, DAC)可以让项目空间管理员或对象创建者能自主决定将其所拥有的项目空间对象（如 table, function, resource 等）授权给其他 ODPS 用户。

本节主要介绍项目空间的用户、角色及授权管理。它主要适用于如下场景：假设你是项目空间的 owner 或管理员，如果有其他人需要申请访问你的项目空间资源，并且这个申请人是属于你的项目团队。那么你可以采用本节介绍的方法来进行用户和授权管理。

如果申请人不属于你的项目团队，那么建议你使用跨项目空间的资源分享功能。

跨项目空间的资源分享

2、项目空间

项目空间(Project)是 ODPS 实现多租户体系的基础，是用户管理数据和计算的基本单位，也是计量和计费的主体。当你申请创建一个项目空间之后，你就是这个空间的 Owner。也就是说，这个项目空间内的所有东西(eg, Table, Resource, Job, Instance, Function)都是你的。注意，都是你的。这就是说，除了你之外，任何人（注意，ODPS 没有超级管理员）都无权访问你的项目空间内的东西，除非有你的授权许可。

3、项目空间的用户管理

当你决定对一个用户授权时，你需要先将该用户添加到你的项目空间中来。只有添加到项目空间中的用户才能够被授权。添加用户时，首先需要知道用户的账号类型，是云账号，还是域账号还是淘宝账号，而且项目空间需要支持相应的账号类型。如果不支持的话，则需要先添加对账号类型的支持。账号类型的相关操作命令如下：

```
LIST AccountProviders; --查看当前 project 支持的账号类型

ADD AccountProvider <apName>; --添加对某种账号类型的支持，有效的 apName 为：ALIYUN, DOMAIN, TAOBAO

REMOVE AccountProvider <apName>; --取消支持某种账号类型
```

Note

- 执行 REMOVE ACCOUNTPROVIDER 命令时，如果当前项目空间中还存在该账号类型的用户，则该命令执行失败。需要先移除当前项目空间下对应该账号类型的所有用户，之后才能执行该操作。

添加用户的命令如下：

```
ADD USER <full_username>; --在项目空间中添加用户
```

Note

- <full_username>的格式：<apName>$<username>
- 云账号的<username>是在 www.aliyun.com 上注册过的有效 email 地址
- 淘宝账号的<username>可以是淘宝会员名（即 taobaonick），也可以是已绑定的手机号码，也可以是已绑定的 email 地址
- 域账号的<username>可以是用户名（如 jun.li），也可以是邮箱地址（如 jun.li@alibaba-inc.com）
- 如果当前项目空间仅支持一种账号类型，那么<full_username>的格式可以简化为<username>，ODPS 会自动添加前缀”<apName>$”

当一个用户离开你的项目团队时，你需要将该用户从你的项目空间中移除。用户一旦从项目空间中被移除，该用户将不再拥有任何访问项目空间资源的权限。移除用户的命令如下：

```
REMOVE USER <full_username>; --在项目空间中移除用户
```

Note

- 当一个用户被移除后，该用户不再拥有访问该项目空间资源的任何权限。
- 当一个用户被移除后，用户与角色的指派关系将被删除，但与该用户有关的 ACL 和 Policy 授权数据仍然会被保留。一旦该用户以后被再添加到该项目空间时，该用户的历史的 ACL 和 Policy 访问权限将被重新激活。
- ODPS 目前不支持在项目空间中彻底移除一个用户及其所有权限数据。

4、项目空间的角色管理

角色(Role)是一组访问权限的集合。当需要对一组用户赋予相同的权限时，可以使用角色来授权。基于角色的授权可以大大简化授权流程，降低授权管理成本。当需要对用户授权时，应当优先考虑是否应该使用角色来完成。

每一个项目空间在创建时，会自动创建一个 admin 的角色，并且为该角色授予了确定的权限：能访问项目空间内的所有对象，能进行用户与角色管理，能对用户或角色进行授权。与 project owner 相比，admin 角色不能将 admin 指派给用户，不能设定项目空间的安全配置，不能修改项目空间的鉴权模型。admin 角色所对应的权限不能被修改。

角色管理相关命令如下：

```
CREATE ROLE <roleName>; --创建角色

DROP ROLE <roleName>; --删除角色

GRANT <roleName> TO <full_username>; --给用户指派某种角色

REVOKE <roleName> FROM <full_username>; --撤销角色指派
```

Note

- 删除一个角色时，ODPS 会检查该角色是否还存在角色指派。若存在，则删除该角色失败。只有在该角色的所有角色指派都被撤销时，删除角色才会成功。

5、对用户或角色进行授权

授权操作一般涉及到三个要素：主体(Subject)，客体(Object)和操作(Action)。在 ODPS 中，主体是指用户或角色，客体是指项目空间中的各种类型对象，操作则与特定对象类型有关，不同类型的对象支持的操作也不尽相同。

ODPS 项目空间支持如下的对象类型及操作：

对象类型	支持的操作	说明
Project	• Read • Write • List • CreateTable • CreateInstance • CreateFunction • CreateResource • CreateJob	• 查看项目空间自身（不包括项目空间内的任何对象）的相关信息，如 CreateTime, Comments 等 • 更新项目空间自身（不包括项目空间内的任何对象）的相关信息，如 Comments • 查看项目空间所有类型的对象列表 • 在项目空间中创建 Table • 在项目空间中创建 Instance • 在项目空间中创建 Function • 在项目空间中创建 Resource • 在项目空间中创建 Job
Table	• Describe • Select • Alter • Update • Drop	• 读取 table 的 Metadata • 读取 table 的 Rows • 修改 table 的 Metadata • 覆盖或添加 table 的 Rows • 删除 table
Function	• Read • Write • Delete • Execute	• 读取 • 更新 • 删除 • 运行
Resource Instance Job	• Read • Write • Delete	• 读取 • 更新 • 删除

在添加用户或创建角色之后，需要对用户或角色进行授权。ODPS 支持两种授权机制来完成对用户或角色的授权：ACL 授权和 Policy 授权。

ACL 授权是一种基于对象的授权。通过 ACL 授权的权限数据（即访问控制列表, Access Control List）被看做是该对象的一种子资源。只有当对象已经存在时，才能进行 ACL 授权操作；当对象被删除时，通过 ACL 授权的权限数据会被自动删除。ACL 授权支持类似于 SQL92 定义的 GRANT/REVOKE 语法，它通过简单的授权语句来完成对已存在的项目空间对象的授权或撤销授权。

Policy 授权则是一种基于主体的授权。通过 Policy 授权的权限数据（即访问策略）被看做是授权主体的一种子资源。只有当主体（用户或角色）存在时，才能进行 Policy 授权操作；当主体被删除时，通过 Policy 授权的权限数据会被自动删除。Policy 授权使用 ODPS 自定义的一种访问策略语言来进行授权，允许或禁止主体对项目空间对象的访问权限。

下面分别介绍这两种授权机制。

6、ACL 授权

这是一种比较经典的授权模型，授权简单、容易使用。ODPS 支持的 ACL 授权方法是采用类似 SQL92 定义的 GRANT/REVOKE 语法来进行授权。ACL 授权语法如下：

```
GRANT privileges ON project_object TO project_subject

REVOKE privileges ON project_object FROM project_subject

privileges ::= action_item1, action_item2, ...

project_object ::= PROJECT project_name | TABLE schema_name |

                   INSTANCE inst_name | FUNCTION func_name |

                   RESOURCE res_name | JOB job_name

project_subject ::= USER full_username | ROLE role_name
```

熟悉 SQL92 定义的 GRANT/REVOKE 语法或者熟悉 Oracle 数据库安全管理的用户容易发现，ODPS 的 ACL 授权语法并不支持 [WITH GRANT OPTION] 授权参数。也就是说，当用户 A 授权用户 B 访问某个对象时，用户 B 无法将权限进一步授权给用户 C。那么，所有的授权操作都必须由具有以下三种身份之一的用户来完成：

- 项目空间 Owner
- 项目空间中拥有 admin 角色的用户

- 项目空间中对象创建者

下面给出一个简单的使用 ACL 授权的应用实例：

场景说明：云账号用户 alice@gmail.com 和 bob@gmail.com 是新加入到项目空间 prj1 的成员。在 prj1 中，他们需要提交作业、创建数据表、查看项目空间已存在的对象。

管理员执行的授权操作如下：

```
USE prj1; --打开项目空间

ADD USER ALIYUN$alice@gmail.com; --添加用户

ADD USER ALIYUN$bob@gmail.com; --添加用户

CREATE ROLE worker; --创建角色

GRANT worker TO ALIYUN$alice@gmail.com; --角色指派

GRANT worker TO ALIYUN$bob@gmail.com; --角色指派

GRANT CreateInstance, CreateResource, CreateFunction, CreateTable, List ON PROJECT prj1 TO ROLE worker; --对角色授权
```

7、Policy 授权

Policy 授权是一种新的授权机制,它主要解决 ACL 授权机制无法解决的一些复杂授权场景，比如：

- 一次操作对一组对象进行授权，如所有的函数、所有以 “taobao” 开头的表。
- 带限制条件的授权，如授权只会在指定的时段内才会生效、当请求者从指定的 IP 地址发起请求时授权才会生效、或者只允许用户使用 SQL(而不允许其它类型的 Task)来访问某张表。

Policy 授权机制使用访问策略语言(Access Policy)来描述授权，关于访问策略语言的详细描述，请参考如下章节：

访问策略语言

Policy 授权语句格式如下：

```
GET POLICY; --读取项目空间的 Policy

PUT POLICY <policyFile>; --设置（覆盖）项目空间的 Policy

GET POLICY ON ROLE <roleName>; --读取项目空间中某个角色的 Policy

PUT POLICY <policyFile> ON ROLE <roleName>; --设置（覆盖）项目空间中某个角色的 Policy
```

下面给出一个简单的使用 Policy 授权的应用实例：

场景说明：授权用户 alice@gmail.com 只能在”2013-11-11T23:59:59Z”这个时间点之前、只能从”10.32.180.0/23”这个 IP 段提交请求，只允许在项目空间 prj1 中执行 CreateInstance, CreateTable 和 List 操作，禁止删除 prj1 下的任何 table。

编写 Policy 如下：

```
{

"Version": "1",

"Statement":

 [{

    "Effect":"Allow",

    "Principal":"ALIYUN$alice@gmail.com",

    "Action":["odps:CreateTable","odps:CreateInstance","odps:List"],

    "Resource":"acs:odps:*:projects/prj1",

    "Condition":{

        "DateLessThan": {

            "acs:CurrentTime":"2013-11-11T23:59:59Z"

        },
```

```
        "IpAddress": {
            "acs:SourceIp":"10.32.180.0/23"
        }
    }
},
{
    "Effect":"Deny",
    "Principal":"ALIYUN$alice@gmail.com",
    "Action":"odps:Drop",
    "Resource":"acs:odps:*:projects/prj1/tables/*"
}]
}
```

Note

- 目前仅支持 Role Policy 和 Project Policy，暂不支持 User Policy。
- 每种 Policy 只支持一个 Policy 文件。由于 Put Policy 操作会覆盖已有的 Policy，所以执行设置 Policy 操作时，应按如下顺序操作：Get Policy -> Merge Policy Statements -> Put Policy.

8、ACL 授权 v.s Policy 授权

ACL 授权的特点：

1. 授权或撤销授权时，要求 Grantee(如 user 或 role)和 Object(如 table)必须已经存在。这点与 Oracle 授权特性相似，可以避免“删除并重建同名对象”所带来的安全风险。
2. 删除一个对象时，自动撤销与该对象关联的所有授权。
3. 仅支持 Allow（白名单）授权，不支持 Deny（黑名单）授权。
4. 使用经典的 Grant/Revoke 授权语法进行授权。语法简单，使用时不易出错。

5. 不支持带限制条件的授权。
6. 适合于简单的授权需求：授权不带限制条件，不需要 Deny，并只需要对已存在对象进行授权。

Policy 授权的特点：

1. 授权或撤销授权时，不关心 Grantee 或 Object 存在与否。授权对象可以支持以通配符“*”来表达，比如，“projects/tbproj/tables/taobao*”，它表示项目空间 tbproj 中所有以“taobao”开头的表。这点与 Mysql 授权特性相似，它允许对不存在的对象授权，授权者应考虑到“删除并重建同名对象”所带来的安全风险。
2. 删除一个对象时，与该对象关联的 Policy 授权不会被删除。
3. 同时支持 Allow（白名单）和 Deny（黑名单）授权。当 Allow 和 Deny 授权同时存在时，遵循 Deny 优先原则。
4. 支持带限制条件的授权。授权者可以对 Allow 或 Deny 授权施加条件限制（目前支持 20 种条件操作）。比如，允许请求者的 IP 为指定的 IP 地址范围，同时访问时间必须在 2012-11-11T23:59:59Z 之前。
5. 适合于相对复杂的授权需求：带授权限制条件，有 Deny 授权需求，希望支持对“未来的对象”授权。

9、查看权限

ODPS 支持从多种维度查看权限，具体包括查看指定用户的权限、查看指定角色的权限、以及查看指定对象的授权列表。

在展现用户权限或角色权限时，ODPS 使用了如下的标记字符：A、C、D、G，它们的含义如下：

- A: 表示 Allow，即允许访问。
- D: 表示 Deny，即拒绝访问。
- C: 表示 with Condition，即为带条件的授权，只出现在 policy 授权体系中。具体的条件需要用户查看相关的 policy。
- G: 表示 with Grant option，即可以对 object 进行授权。

一个简单的展现权限的样例如下：

```
odps@securitytest> show grants for odpstest1;

[roles]

dev
```

```
Authorization Type: ACL

[role/dev]

A       projects/securitytest/tables/t1: Select

[user/odpstest1@aliyun.com]

A       projects/securitytest: CreateTable | CreateInstance | CreateFunction
| List

A       projects/securitytest/tables/t1: Describe | Select



Authorization Type: Policy

[role/dev]

AC      projects/securitytest/tables/test_*: Describe

DC      projects/securitytest/tables/alifinance_*: Select

[user/odpstest1@aliyun.com]

A       projects/securitytest: Create* | List

AC      projects/securitytest/tables/alipay_*: Describe | Select



Authorization Type: ObjectCreator

AG      projects/securitytest/tables/t6: All

AG      projects/securitytest/tables/t7: All
```

10、查看指定用户的权限

```
SHOW GRANTS; --查看当前用户自己的访问权限

SHOW GRANTS FOR <username>; --查看指定用户的访问权限,仅由ProjectOwner和Admin才能有执行权限。
```

11、查看指定角色的权限

```
DESCRIBE ROLE <rolename>; --查看指定角色的访问权限角色指派
```

12、查看指定对象的授权列表

```
SHOW ACL FOR <objectName> [ON TYPE <objectType>]; --查看指定对象上的用户和角色授权列表
```

Note

- 当省略[ON TYPE <objectType>]时，默认的 TYPE 为 Table。

（四）访问策略语言

1、基本术语

权限(Permission)是访问控制的一个基本概念，它表示允许或拒绝一个请求者(Requester)对资源(Resource)执行某种操作(Action)。我们用语句(Statement)来表示单个权限的形式化描述，而用策略(Policy)来表示语句的集合。

访问策略(Access Policy)主要包括如下的访问控制元素：主体(Principal)、操作(Action)、资源(Resource)、访问限制(Access Restriction)和效果(Effect)。下面分别对这几个基本实体进行简要描述。

1.1 主体(Principal)

主体(Principal)是指访问策略中的权限被指派的对象。比如，访问策略“允许张三在 2011 年 12 月 31 日之前对资源 SampleBucket 执行 CreateObject 操作”中的主体是“张三”。

1.2 操作(Action)

操作(Action)是指主体对资源的访问方法。比如，访问策略“允许张三在 2011 年 12 月 31 日之前对资源 SampleBucket 执行 CreateObject 操作”中的操作是“CreateObject”。

1.3 资源(Resource)

资源(Resource)是指主体请求访问的对象。比如，访问策略“允许张三在 2011 年 12 月 31 日之前对资源 SampleBucket 执行 CreateObject 操作”中的资源是“SampleBucket”。

1.4 访问限制(Access Restriction)

访问限制(Access Restriction)是指权限生效的限制条件。比如，访问策略“允许张三在 2011 年 12 月 31 日之前对资源 SampleBucket 执行 CreateObject 操作”中的限制条件是“在 2011 年 12 月 31 日之前”。

1.5 效力(Effect)

授权效力包括两个方面：允许操作（Allow）和拒绝操作（Deny）。通常，Deny 有更高的效力，在权限检查时会优先使用。注意，拒绝操作和“撤销授权”是完全独立的两个概念，撤销授权通常包括撤销对 Allow 和 Deny 这两种不同效力的授权，比如传统数据库一般支持 Revoke 和 Revoke Deny 两种操作。

2、访问策略语言结构

下图描述了访问策略的基本结构。一个策略(Policy)由以下部分组成：(1) 可选的策略头部元素；(2) 一条或多条语句(Statement)。策略头部元素是可选的，它主要包括策略的版本信息。策略的主体是语句的集合。

```
|            +-----------+        |
|                                  |
|            +-----------+        |
|            | Statement |        |
|            +-----------+        |
|                                  |
|            +-----------+        |
|            | Statement |        |
|            +-----------+        |
|                                  |
+----------------------------------+
```

Statement 结构：

```
 +-----------------+
 | Effect          |
 +-----------------+
 | Principal       |
 +-----------------+
 | Action          |
 +-----------------+
 | Resource        |
```

```
+-----------------+

| Condition Block |

+-----------------+
```

2.1 授权语句(Statement)结构

一条授权语句(Statement)包括如下条目：

- Effect: 指明该条语句的权限类型，取值必须为 Allow 或 Deny。
- Principal: 如果 Policy 在授权时是与用户或角色绑定，那么 Principal 条目可省略该项。如果 Policy 在授权时是与项目空间或项目空间内的对象绑定，那么该项便不能省略。
- Action: 它表示授权操作，可以是一个或多个操作名，可支持通配符号“*”和“?”。例如，Action = “*” 表示所有的操作。
- Resource: 它表示授权对象，可以是一个或多个对象名，可支持通配符号“*”和“?”。例如 Resource = “*” 表示所有的对象。
- Condition Block: 条件块是该条授权语句所述权限得以生效的条件。条件块结构请参见下节的描述。

2.2 条件块结构

条件块由一个或多个条件子句构成。一个条件子句由条件操作类型、条件关键字和条件值组成。条件操作类型和条件关键字在下文中会有详细描述。

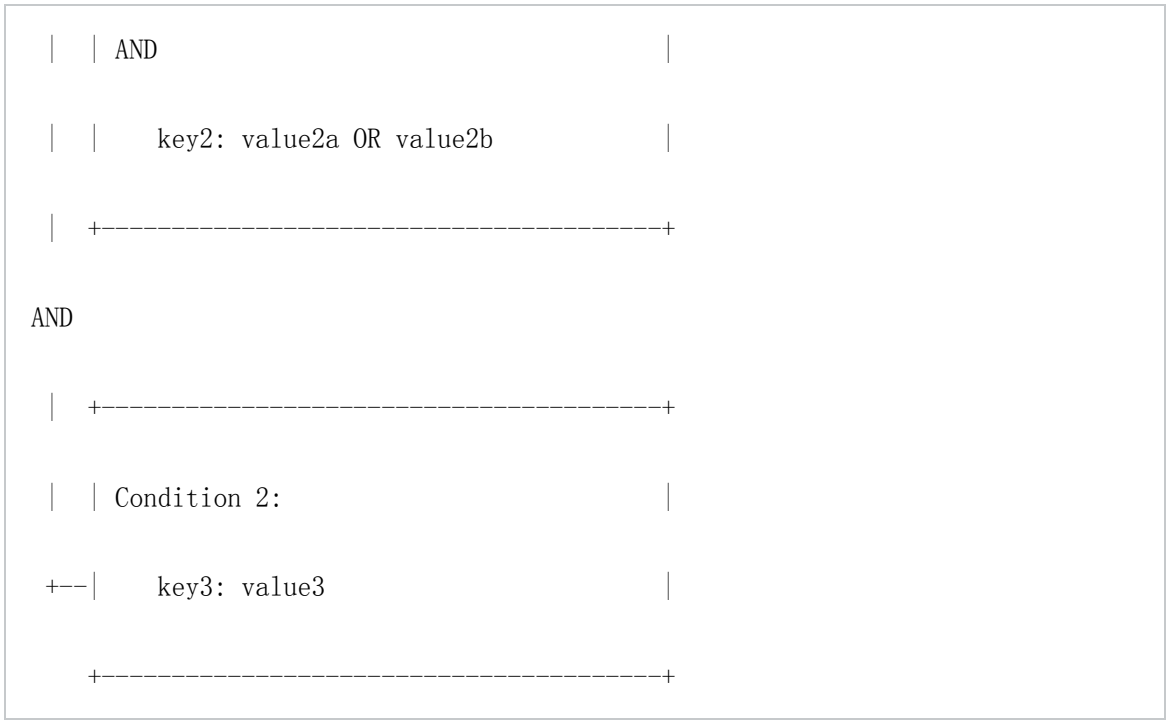
条件块的满足性判断原则如下图所示。描述如下：(1) 一个条件关键字可以指定一个或多个值，在条件检查时，如果条件关键字的值与指定值中的某一个相等，即可判定条件满足。(2) 同一种条件操作类型的条件子句下的多个条件关键字同时满足的情况下，才能判定该条件子句满足。(3) 条件块下的所有条件子句同时满足的情况下，才能判定该条件块满足。

```
条件块判断逻辑

 +-------------------------------------+

    | Condition 1:                      |

+--|    key1: value1a OR value1b OR value1c |
```



2.3、条件操作类型

我们支持如下条件操作类型：字符串类型、数字类型、日期类型、布尔类型和 IP 地址类型。每种条件操作类型分别支持如下的方法：

String

- StringEquals
- StringNotEquals
- StringEqualsIgnoreCase
- StringNotEqualsIgnoreCase
- StringLike
- StringNotLike

Numeric

- NumericEquals
- NumericEquals
- NumericLessThan
- NumericLessThanEquals
- NumericGreaterThan
- NumericGreaterThanEquals

Date and time

- DateEquals
- DateNotEquals
- DateLessThan
- DateLessThanEquals
- DateGreaterThan
- DateGreaterThanEquals

Boolean

- Bool

IP address

- IpAddress
- NotIpAddress

2.4 条件关键字

首先，ODPS 支持开放云服务 ACS (Aliyun Cloud Service) 保留的如下条件关键字：

ACS 保留条件关键字	类型	说明
acs:CurrentTime	Date and time	Web Server 接收到请求的时间，以 ISO 8601 格式表示，如 2012-11-11T23:59:59Z
acs:SecureTransport	Boolean	发送请求是否使用了安全信道，如 HTTPS
acs:SourceIp	IP address	发送请求时的客户端 IP 地址
acs:UserAgent	String	发送请求时的客户端 UserAgent
acs:Referer	String	发送请求时的 HTTP referer

此外，ODPS 新增了如下的条件关键字：

ODPS 扩展条件关键字	类型	说明
odps:TaskType	String	ODPS 任务类型。目前支持的任务类型有：SQL, MR, DT, XLIB
dt:SourceType	String	ODPS DT Task 数据源类型。目前支持的数据源类型有：odps, file, oracle, mysql
dt:SourceIp	IP address	ODPS DT Task 数据源 IP 地址
dt:SourceResource	String	ODPS DT Task 数据源对象，参考 Resource 命名规范
dt:TargetType	String	ODPS DT Task 目标类型。目前支持的目标类型有：odps, file, oracle, mysql
dt:TargetIp	IP address	ODPS DT Task 目标 IP 地址
dt:TargetResource	String	ODPS DT Task 目标对象，参考 Resource 命名规

ODPS 扩展条件关键字	类型	说明
		范

3、访问策略语言规范

3.1Principal 命名规范

Principal 是指请求发送者的身份。目前仅支持阿里云帐号、域账号和淘宝账号的身份表示。对于云账号，可以支持 ID 或 DisplayName 的表示。比如：

"Principal": "43274"
"Principal": "ALIYUN$jun.li@aliyun.com"
"Principal": ["ALIYUN$jun.li@aliyun.com","DOMAIN$jack@alibaba-inc.com", "TAOBAO$alice"]

3.2Resource 命名规范

我们使用如下格式来命名 odps 所提供的资源。格式如下：
acs:<service-name>:<namespace>:<relative-id>

Note

- acs: 保留的 Resource 头部。
- service-name: 阿里云开放服务名称，如 odps, oss, ots 等。
- namespace: 命名空间，用于资源隔离。如果以云帐号 ID 来做资源隔离，那么可取值为云帐号 ID。如果不支持该项，可以使用通配符“*”号来代替。
- relative-id: 与 service 相关的资源描述部分，其语义由具体 service 指定。这部分的格式描述支持类似于文件路径的树状结构。以 odps 为例，relative-id 的格式为：projects/<project_name>/<object_type>/<object_name>。

ODPS Resource 命名样例:

Resource 命名实例	说明
*	项目空间中的所有对象
projects/prj1/tables/t1	项目空间 prj1 中的 t1 表
projects/prj1/instances/*	项目空间 prj1 中的所有的 instances
projects/prj1/tables/*	项目空间 prj1 中的所有表
projects/prj1/tables/taobao* | 项目空间 prj1 中的所有以 taobao 开头的表	

3.3Action 命名规范

Action 命名规范如下：<service-name>:<action-name>

Note

- service-name: 开放服务名称，如 odps, oss, ots 等。
- action-name: service 相关的操作接口名称。

ODPS Action 命名样例:

Action 命名实例	说明
*	所有操作
odps:*	ODPS 的所有操作
odps:CreateTable	ODPS 的 CreateTable 操作
odps:Create*	ODPS 的所有以 Create 开头的操作

3.4Condition Keys 命名规范

1、开放云服务 ACS 保留的条件关键字命名格式为：acs:<condition-key>

Note
- condition-key: ACS 保留了 5 种条件关键字，所有的开放服务可以共用。它们是：acs:CurrentTime, acs:SecureTransport, acs:SourceIp, acs:UserAgent, acs:Referer.

2、具体服务相关的条件关键字的命名格式为：<service-name>:<condition-key>

Note
- condition-key: service 自定义的条件关键字。

3.5 访问策略样例

Policy Sample:

```
{
"Version": "1",
"Statement":
 [{
    "Effect":"Allow",
    "Principal":"ALIYUN$alice@gmail.com",
    "Action":["odps:CreateTable","odps:CreateInstance","odps:List"],
    "Resource":"acs:odps:*:projects/prj1",
    "Condition":{
        "DateLessThan": {
            "acs:CurrentTime":"2013-11-11T23:59:59Z"
        },
        "IpAddress": {
            "acs:SourceIp":"10.32.180.0/23"
        }
    }
```

```
},
{
    "Effect":"Deny",
    "Principal":"ALIYUN$alice@gmail.com",
    "Action":"odps:Drop",
    "Resource":"acs:odps:*:projects/prj1/tables/*"
}]
}
```

解释：授权用户 alice@gmail.com 只能在”2013-11-11T23:59:59Z”这个时间点之前、只能从”10.32.180.0/23”这个 IP 段提交请求，只允许在项目空间 prj1 中执行 CreateInstance, CreateTable 和 List 操作，禁止删除 prj1 下的任何 table。

（五）快速开始

ODPS 的各项功能都可以通过 Console 来访问，以下是一个使用 Console 运行 ODPS 处理数据的快速示例。

首先从 www.aliyun.com 下载一个 Console 客户端，解压到一个文件夹中。由于 Console 是用 Java 开发的，因此要确保机器上有 JRE 1.6 的环境。在解压的文件夹中可以看到如下 4 个文件夹：

```
bin/  conf/  lib/  plugins/
```

在 conf 文件夹中有 odps_conf.ini 文件。编辑此文件，填写相关信息：

```
project_name=${your_project_name}
    #指定您想进入的项目空间。此项可忽略，缺省时进入 console 后需要使用"use project"命令进入您的项目空间。
access_id=
access_key=
    #这两项是由云账号提供的，在阿里云官网上可以获取
end_point=http://service.odps.aliyun.com/api
    #链接 ODPS 使用的 URL
```

修改好配置文件后运行 bin 目录下的 odpscmd(在 LINUX 系统下是./bin/odpscmd，WINDOWS 下运行./bin/odpscmd.bat)，现在可以运行几个 SQL。

```
CREATE TABLE tbl1(id BIGINT);
INSERT OVERWRITE TABLE tbl1 SELECT COUNT(*) FROM tbl1;
SELECT 'WELCOME TO ODPS!' FROM tbl1;
```

ODPS 同样提供了上传下载数据的方式，下面我们将给出这样一个示例：

创建销售表 sale_detail，

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

此表是一张分区表，详细介绍请参阅 *创建表(CREATE TABLE)* 。向表中添加杭州和上海两地区 2013 年 10 月和 11 月份的分区：

```
ALTER TABLE sale_detail ADD PARTITION (sale_date='201310', region='hangzhou');
ALTER TABLE sale_detail ADD PARTITION (sale_date='201311', region='hangzhou');
```

```
ALTER TABLE sale_detail ADD PARTITION (sale_date='201310', region='shanghai');
ALTER TABLE sale_detail ADD PARTITION (sale_date='201311', region='shanghai');
```

添加分区的介绍请参阅 *添加分区(ADD PARTITIONS)* 。准备这 4 个分区的数据：杭州市 2013 年 10 月销售记录：

shop1,1,100

shop2,2,200

杭州市 2013 年 11 月销售记录：

shop3,3,300

shop4,4,400

上海市 2013 年 10 月销售记录：

shop5,5,500

shop6,6,600

上海市 2013 年 11 月销售记录：

shop7,7,700

shop8,8,800

将这 4 份数据保存到 4 分本地文件中：hangzhou_10.txt, hangzhou_11.txt, shanghai_10.txt, shanghai_11.txt。分别将其上传至 ODPS 的表中：

```
UPLOAD sale_detail PARTITION(sale_date='201310',region='hangzhou') FROM hangzhou_10.txt;
UPLOAD sale_detail PARTITION(sale_date='201311',region='hangzhou') FROM hangzhou_11.txt;
UPLOAD sale_detail PARTITION(sale_date='201310',region='shanghai') FROM shanghai_10.txt;
UPLOAD sale_detail PARTITION(sale_date='201311',region='shanghai') FROM shanghai_11.txt;
```

验证数据：

```
SELECT * FROM sale_detail;
```

五、公测须知

1、申请条件

- 为了您更好的感受到 ODPS 带给您的极致体验，我们建议拥有较大规模数据的用户来体验 ODPS 服务。如：百 G 数据规模以上的用户。
- 由于 ODPS 对用户的技术能力有一定要求，建议您有 SQL、hadoop、java 等的技术背景。
- 目前我们暂时对企业用户开放，个人暂不开放.

2、申请流程

第一步：用户登录 ODPS 产品详情页：

http://www.aliyun.com/product/odps/

第二步：点击屏幕右上角“获取邀请码”按钮

第三步：填写用户信息后点击提交（提交成功后等待阿里云审核）

第四步:登录绑定阿里云账号的邮箱查看审核是否通过(审核通过后,邮件中会显示邀请码)
第五步：登录 ODPS 控制台输入邀请码后开始体验

3、审核周期

审核周期为：3 个工作日

4、公测时间

自 2014 年 1 月 25 日凌晨零时起，至 2014 年 3 月 25 日 24 时结束

六、技术分享

1、ODPSSQL、Hive 和 Mysql 的对比

由于很多ODPS用户可能已经对传统数据库SQL或Hive很熟悉，所以这里我们通过以下表格，简单说明ODPSSQL、Hive和Mysql（作为传统关系数据库代表）之间的一些异同，便于这些用户能够更轻松快速了解ODPSSQL。

	MySQL	Hive	ODPSSQL
查询语言	SQL	HQL（类似 SQL 语法）	ODPSSQL（和 HQL 很接近但有些区别，类似 SQL 语法）
主要存储方式	硬盘	HDFS（可扩展，支持多种存储方式）	ODPS 表
数据规模	较低	高	高
执行方式	引擎	MapReduce 分布式处理	MapReduce 分布式处理
执行延迟	低	高	较高
索引	有	有（基本无用）	无
Partition（分区）	有	有	有