

深入阿里云产品
开放数据处理服务 ODPS
学生实验手册

<课程编号> ACA21104
<版本号> V1
<日期>20150825



Copyright©2014，阿里巴巴版权所有
免责声明

保留权利通知

目录

一、实验目标.....	4
1.1 实验背景介绍.....	4
1.2 实验环境架构.....	4
1.3 云端服务环境.....	4
1.4 云端开发桌面环境.....	4
二、实验内容.....	5
2.1 实验：检查本地系统软件安装配置情况.....	5
2.2 使用老版客户端.....	5
2.2.1 实验：安装配置老版客户端.....	5
2.2.2 实验：交互界面执行常用命令.....	8
2.2.3 实验：使用 -f 参数执行指定文件中的命令集.....	9
2.3 新版客户端简单入门.....	10
2.3.1 实验：安装配置新版客户端.....	10
2.3.2 实验：交互界面执行常用命令.....	10
2.3.3 实验：使用 -f 参数执行指定文件中的命令集.....	11
2.3.4 实验：使用 -e 参数执行命令集.....	12
2.4 tunnel 数据传输命令基本操作.....	12
2.4.1 实验： tunnel upload 目标为单表：.....	12
2.4.2 实验： tunnel upload 目标为分区表：.....	13
2.4.3 实验： tunnel upload 源文件为目录：.....	13
2.4.4 实验： tunnel upload 容忍错误记录：.....	13
2.4.5 实验： tunnel upload 扫描文件：.....	14
2.4.6 实验： tunnel upload 行、列分隔符：.....	14
2.4.7 实验： tunnel upload 多线程：.....	14
2.4.8 实验： tunnel download 非分区表：.....	15
2.4.9 实验： tunnel download 分区表：.....	15
2.5 dship 数据传输命令基本操作.....	15
2.5.1 实验： 安装配置 dship：.....	15
2.4.1 实验： dship upload 目标为单表：.....	16
2.5.2 实验： tunnel upload 目标为分区表：.....	16
2.5.3 实验： dship download 非分区表：.....	17
2.5.4 实验： dship download 分区表：.....	17
2.6 tunnel JAVA SDK 定制开发数据传输服务.....	17
2.6.1 实验：安装配置 eclipse 开发环境：.....	17
2.6.2 实验：单线程上传文件：.....	19
2.6.3 实验：单线程下载文件：.....	29
2.7 ODPS SQL 运算符.....	32
2.7.1 实验：建立测试用表，并加载数据：.....	32
2.7.2 实验：使用关系型运算符：.....	33
2.7.3 实验：使用算术运算符：.....	34
2.7.4 实验：使用位运算符：.....	34
2.7.5 实验：使用逻辑运算符：.....	34

2.7.6 实验：类型的显示转换：	35
2.7.7 实验：类型的隐式转换：	35
2.8 ODPS DDL 基本操作	35
2.8.1 实验：使用 SQL 建表：	35
2.8.2 实验：分区操作：	36
2.8.3 实验：修改表属性：	37
2.8.4 实验：视图操作：	38
2.9 ODPS DML 基本操作	38
2.9.1 实验：建表并准备数据：	38
2.9.2 实验：查询操作	38
2.9.3 实验：更新数据：	39
2.9.4 实验：多路输出：	40
2.9.5 实验：动态分区：	41
2.9.6 实验：join 以及 mapjoin HINT：	41
2.9.7 实验：子查询：	43
2.9.8 实验：UNION ALL：	44
2.9.9 实验：CASE WHEN 表达式：	44
2.10 内置函数	45
2.10.1 实验：数值类函数：	45
2.10.2 实验：字符串类函数：	46
2.10.3 实验：日期类函数：	47
2.10.4 实验：窗口函数：	49
2.10.5 实验：聚合函数：	50
2.10.6 实验：其他函数：	51
2.11 执行计划和 Logviewer	52
2.12 用户自定义函数 UDF	53
2.12.1 实验：搭建配置实验环境：	53
2.12.2 实验：最简单的 UDF 示例 LowCase	53
2.12.3 实验：UDF 之最简单的加密	60
2.12.4 实验：UDTF 之多值列拆分：散伙分行李	61
2.12.5 实验：UDAF 之变异系数	61
2.13 MapReduce	63
2.13.1 实验：搭建配置实验环境：	63
2.13.2 实验：Word Count, MR 界的 “Hello World! ”	64
2.13.3 实验：好友推荐	71
2.13.4 实验：血缘分析	78
2.14 安全与权限	83
2.14.1 实验：准备测试环境	83
2.14.2 实验：A 用户授权给 B 用户	83
2.14.3 实验：角色管理与授权	84
2.14.4 实验：鉴权模型查看与管理	86
2.14.5 实验：基于标签的安全控制	87
2.14.6 实验：跨项目空间的资源分享	89

2.14.7 实验：项目空间保护.....	90
2.14.8 实验：访问策略语言.....	91
三、实验总结.....	94

一、实验目标

1.1 实验背景介绍

本实验 Demo 是基于课程“ACA21104 深入阿里云产品-开放数据处理服务 ODPS”中的实验环节而设计，实验结合实际数据和案例，深入浅出的演示了如何使用 ODPS 的各种功能模块去完成数据加载、处理和分析等。

1.2 实验环境架构

实验环境架构：阿里云开放数据处理服务 ODPS

1.3 云端服务环境

暂无

1.4 云端开发桌面环境

二、实验内容

2.1 实验：检查本地系统软件安装配置情况

(1) 查看本次实验课用到的安装介质：

```
dir C:\ODPS_DEMO\InstallMedia
```

至少应该包含以下文件：

```
eclipse-java-luna-SR2-win32-x86_64.zip
odpscmd_public.zip
odps-cli-java.zip
odps-dship.zip
odps-eclipse-plugin-bundle-0.16.0.zip
odps-sdk-core-0.18.3-public.jar
jdk-6u45-windows-x64.exe
npp_V6.8.2_Installer.1440141668.exe
```

(2) 检查系统是否安装了 Java 运行环境（1.6 及以上版本）：

```
java -version
```

(3) 检查是否安装了 eclipse

(4) 检查是否安装了 Notepad++

(5) 使用实验账号，登录阿里云官网，检查账号下的可用资源：

应至少包括 ODPS 服务；

如无项目，请新建一个项目用于本次实验，本实验中使用项目名称为 aca21104_demo；

检查可用 AccessKeyID 和 AccessKeySecret，并记录一对用于后续实验；

2.2 使用老版客户端

2.2.1 实验：安装配置老版客户端

(1) 找到安装介质 odps-cli-java.zip，解压缩到 C:\ODPS_DEMO\

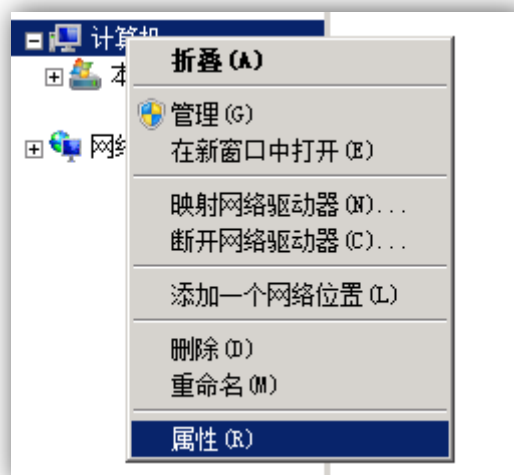
(2) 进入目录 C:\ODPS_DEMO\odps-cli-java\odps\conf\, 打开文件 odps.conf, 修改配置信息:

```
access_id=XXXXXXXXXXXXXXXXXXXX
access_key=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
end_point=http://service.odps.aliyun.com/api
project_name=aca21104_demo
```

(3) 通过命令行, 进入 C:\ODPS_DEMO\odps-cli-java\odps\bin\, 执行 odps, 进入交互界面, 确认安装是否配置成功。

(4) 将 C:\ODPS_DEMO\odps-cli-java\odps\bin 加入环境变量 PATH, 方便通过命令行调用 odps:

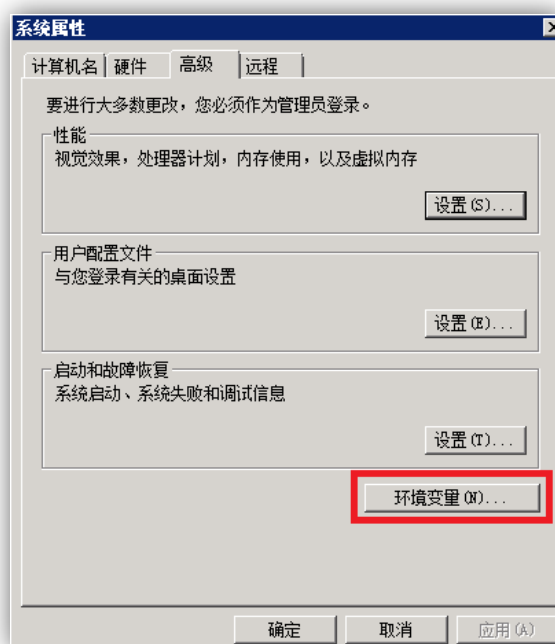
右键点击 *我的电脑*, 弹出菜单中点击 *属性*:



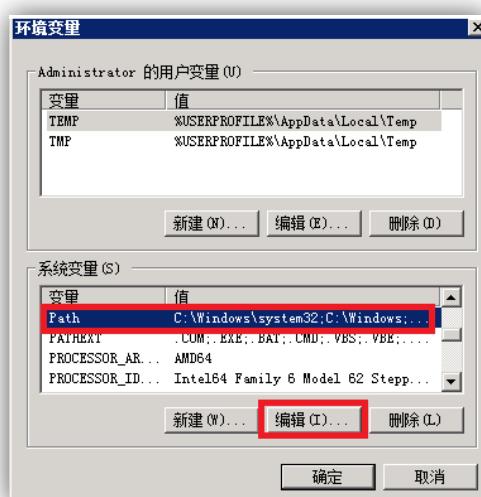
点击 *高级系统设置*:



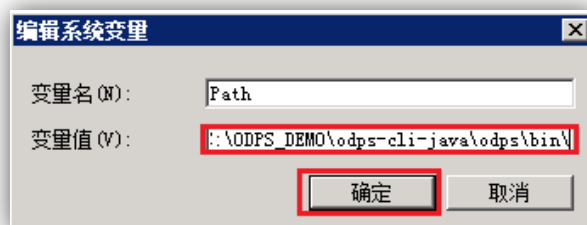
弹出的系统属性窗口中，点击环境变量：



找到系统变量 PATH，点编辑：



将 odps 的可执行命令所在路径加入到 PATH 变量中去，并点击确认：



新打开一个命令行界面，直接执行命令 `odps`，即可进入交互界面。

2.2.2 实验：交互界面执行常用命令

(1) 进入交互界面

```
odps
```

(2) 分别执行下述命令

```
# 查看帮助信息
help
# 切换项目
use aca21104_demo
# 查看当前项目的详细信息
desc project;
# 查看具体命令的帮助信息
help ls
# 列出表信息
ls tables;
# 查看某个表的具体信息
desc <table_name>
# 进入安全管理命令空间
security
# 添加用户 ben.ning@aliyun.com 到当前项目空间中:
add user ALIYUN$ben.ning@aliyun.com
# 退出安全命令空间
quit
# 进入 SQL 命令子空间:
sql
# 新建表 dual
create table dual (x string);
# 插入一条数据
insert into table dual select count(*) from dual;
# 查看表中记录:
select * from dual;
# 退出 SQL 命令空间
quit
```

2.2.3 实验：使用 -f 参数执行指定文件中的命令集

(1) 在目录 C:\ODPS_DEMO\resources\01-BasicKnowledge\ 中的命令文件 crt_tbl.cmd:

```
use aca21104_demo;
sql
drop table if exists t_test;
create table t_test (id int, name string);
insert into table t_test select 1,'odps' from dual;
select * from t_test;
```

- (2) 使用 odps 调用命令文件：

```
odps -f C:\ODPS_DEMO\resources\01-BasicKnowledge\crt_tbl.cmd
```

2.3 新版客户端简单入门

2.3.1 实验：安装配置新版客户端

- (1) 找到安装介质 odpscmd_public.zip，解压缩到 C:\ODPS_DEMO\
(2) 进入目录 C:\ODPS_DEMO\odpscmd_public\conf\，打开文件 odps_config.ini，修改配置信息：

```
project_name=aca21104_demo  
access_id=XXXXXXXXXXXXXXXXXXXX  
access_key=XXXXXXXXXXXXXXXXXXXXXXXXXXXX  
end_point=http://service.odps.aliyun.com/api  
tunnel_endpoint=http://dt.odps.aliyun.com  
log_view_host=http://logview.odps.aliyun.com  
https_check=true
```

- (3) 通过命令行，进入 C:\ODPS_DEMO\odpscmd_public\bin\，执行 odpscmd，进入交互界面，确认安装是否配置成功。
(4) 将 C:\ODPS_DEMO\odpscmd_public\bin 加入环境变量 PATH，方便通过命令行调用 odpscmd

2.3.2 实验：交互界面执行常用命令

- (1) 进入交互界面

```
odpscmd
```

- (2) 分别执行下述命令

```
# 查看帮助信息
help;

# 切换项目
use aca21104_demo;

# 查看当前项目的详细信息
desc project aca21104_demo;

# 列出表信息
ls tables;

# 查看某个表的具体信息
desc dual;

# 查看表中记录数
count dual;
select count(*) from dual;

# 查看表记录内容:
read dual;
select * from dual;

# 添加用户 ben.ning@aliyun.com 到当前项目空间中来:
list users;
remove user ALIYUN$ben.ning@aliyun.com;
add user ALIYUN$ben.ning@aliyun.com;

# 新建表 t_test
create table if not exists t_test (id int, name string);

# 插入一条数据
insert into table t_test select * from t_test;

# 查看表中记录:
read t_test;
```

2.3.3 实验：使用 -f 参数执行指定文件中的命令集

(1) 在目录 C:\ODPS_DEMO\resources\01-BasicKnowledge\ 中的命令文件

crt_tbl_new.cmd:

```
use aca21104_demo;
drop table if exists t_test;
create table t_test (id int, name string);
insert into table t_test select 1,'odps' from dual;
read t_test;
```

(2) 使用 odpscmd 调用命令文件:

```
odpscmd -f C:\ODPS_DEMO\resources\01-BasicKnowledge\crt_tbl_new.cmd
```

(3) 使用 odpscmd 执行命令文件中的一部分命令（跳过开头的命令）：

```
odpscmd -f C:\ODPS_DEMO\resources\01-BasicKnowledge\crt_tbl_new.cmd -k 4
```

2.3.4 实验：使用 -e 参数执行命令集

(1) 使用 odpscmd -e 执行多个命令：

```
odpscmd -e "insert into table t_test select 2, 'odpscmd' from dual; read t_test;"
```

(2) 使用 -e 和 -k 结合在执行命令集时可跳过一些命令：

```
odpscmd -e "insert into table t_test select 2, 'odpscmd' from dual; read t_test;" -k 2
```

2.4 tunnel 数据传输命令基本操作

在目录 C:\ODPS_DEMO\resources\02-DataTransfer\ 中找到文件 crt_tbl_tunnel.sql，执行该脚本，在 ODPS 中创建实验需要的表：

```
odpscmd -f C:\ODPS_DEMO\resources\02-DataTransfer\crt_tbl_tunnel.sql
```

跳转到目录 C:\ODPS_DEMO\resources\02-DataTransfer\，执行 odpscmd，进入交互界面。

2.4.1 实验：tunnel upload 目标为单表：

(1) 使用 upload 子命令，上传文件 people.csv 到表 t_tunnel 中：

```
tunnel upload C:\ODPS_DEMO\resources\02-DataTransfer\people.csv t_tunnel;
```

(2) 检查上传结果：

```
read t_tunnel;
```

(3) 再上传一遍，验证一下 tunnel 上传数据的方式是 append 还是 overwrite：

```
tunnel upload C:\ODPS_DEMO\resources\02-DataTransfer\people.csv t_tunnel;  
read t_tunnel;
```

2.4.2 实验: tunnel upload 目标为分区表:

(1) 创建表 t_tunnel_p 的分区 gender='male' 中:

```
alter table t_tunnel_p add if not exists partition (gender='male');
```

(2) 使用 upload 子命令,上传文件 men.csv 以及 women.csv 到表 t_tunnel_p 的已存在的分区 gender='male' 中:

```
tunnel upload women.csv t_tunnel_p/gender='male';  
tunnel upload men.csv t_tunnel_p/gender='male';
```

(3) 检查表 t_tunnel_p 中的内容,体会一下分区内容和标称值的逻辑关系:

```
read t_tunnel_p;
```

2.4.3 实验: tunnel upload 源文件为目录:

(1) 将目录 C:\ODPS_DEMO\resources\02-DataTransfer\famous_people\ 下的所有文件都上传到 t_tunnel 中去:

```
truncate table t_tunnel;  
tunnel upload famous_people/ t_tunnel;
```

(2) 检查上传结果:

```
read t_tunnel;
```

2.4.4 实验: tunnel upload 容忍错误记录:

(1) 将文件 C:\ODPS_DEMO\resources\02-DataTransfer\people_bad.csv 上传至 t_tunnel:


```
truncate table t_tunnel;  
tunnel upload people_bad.csv t_tunnel;  
read t_tunnel;
```

(2) 指定容忍错误的参数 `-dbr` 为 `true`:

```
truncate table t_tunnel;  
tunnel upload people_bad.csv t_tunnel -dbr true;  
read t_tunnel;
```

2.4.5 实验: tunnel upload 扫描文件:

欲将文件 `C:\ODPS_DEMO\resources\02-DataTransfer\crowd.csv` 上传至 `t_tunnel`, 可以在上传前先做个预扫描:

```
tunnel upload crowd.csv t_tunnel --scan=only;
```

2.4.6 实验: tunnel upload 行、列分隔符:

将文件 `C:\ODPS_DEMO\resources\02-DataTransfer\people_delimiter.csv` 文件上传到 `t_tunnel` 中去:

```
truncate table t_tunnel;  
tunnel upload people_delimiter.csv t_tunnel -fd || -rd &;  
read t_tunnel;
```

2.4.7 实验: tunnel upload 多线程:

使用多个线程将文件 `C:\ODPS_DEMO\resources\02-DataTransfer\crowd.csv` 文件上传到 `t_tunnel` 中去:

```
truncate table t_tunnel;  
tunnel upload crowd.csv t_tunnel -threads 8;  
read t_tunnel;  
count t_tunnel;
```

2.4.8 实验： tunnel download 非分区表：

将表 t_tunnel 下载到本地文件 t_tunnel.csv：

```
tunnel download t_tunnel t_tunnel.csv;
```

2.4.9 实验： tunnel download 分区表：

将表 t_tunnel 下载到本地文件 t_tunnel.csv：

```
tunnel download t_tunnel_p/gender='male' t_tunnel_p.csv;
```

2.5 dship 数据传输命令基本操作

在目录 C:\ODPS_DEMO\resources\02-DataTransfer\ 中找到文件 crt_tbl_dship.sql，执行该脚本，在 ODPS 中创建实验需要的表：

```
odpscmd -f C:\ODPS_DEMO\resources\02-DataTransfer\crt_tbl_dship.sql
```

跳转到目录 C:\ODPS_DEMO\resources\02-DataTransfer\，执行 odpscmd，进入交互界面。

2.5.1 实验： 安装配置 dship：

(1) 安装介质目录中找到 C:\ODPS_DEMO\InstallMedia\odps-dship.zip，解压缩到目录 C:\ODPS_DEMO\odps-dship\dship，找到该目录下的 odps.conf 修改配置：

```
#odps dship config
end_point=http://service.odps.aliyun.com/api
tunnel_endpoint=http://dt.odps.aliyun.com
access_id=XXXXXXXXXXXXXXXXXX
access_key=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
project_name=aca21104_demo
```

(2) 将 dship 的可执行路径添加到系统的环境变量中。

2.4.1 实验： dship upload 目标为单表：

(1) 使用 upload 子命令，上传文件 people.csv 到表 t_dship 中：

```
dship upload C:\ODPS_DEMO\resources\02-DataTransfer\people.csv t_dship
```

(2) 检查上传结果：

```
odps
sql
select * from t_dship;
```

2.5.2 实验： tunnel upload 目标为分区表：

(1) 创建表 t_dship_p 的分区 gender='male' 中：

```
odps
sql
alter table t_dship_p add if not exists partition (gender='male');
```

(2) 使用 upload 子命令，上传文件 men.csv 以及 women.csv 到表 t_dship_p 的已存在的分区 gender='male' 中：

```
dship upload women.csv t_dship_p/gender='male'
dship upload men.csv t_dship_p/gender='male'
```

2.5.3 实验： dship download 非分区表：

将表 t_dship 下载到本地文件 t_dship.csv：

```
dship download t_dship t_dship.csv
```

2.5.4 实验： dship download 分区表：

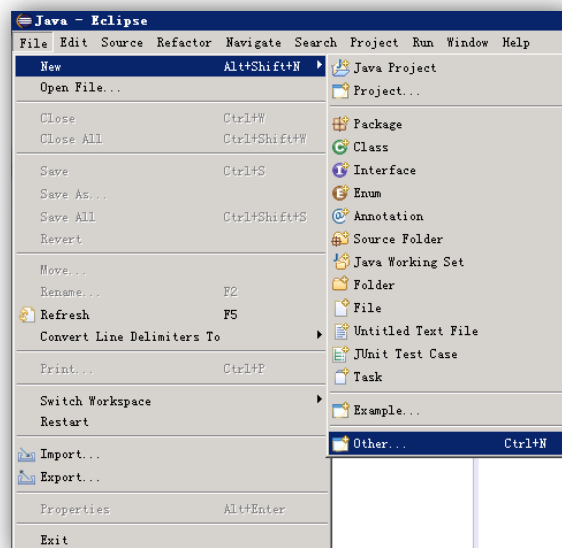
将表 t_dship 下载到本地文件 t_dship.csv：

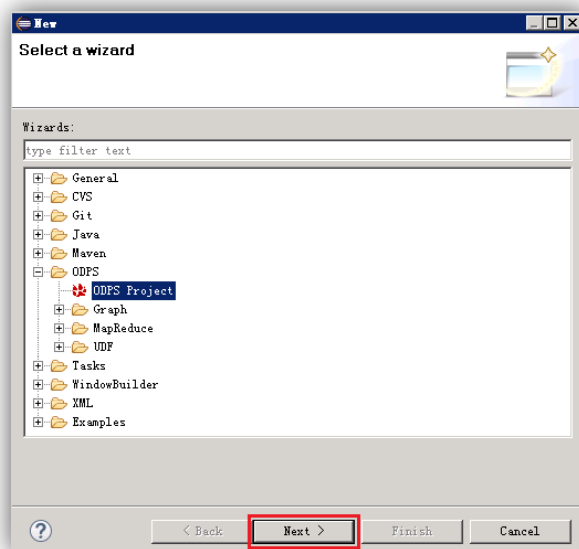
```
dship download t_dship_p/gender='male' t_dship_p.csv
```

2.6 tunnel JAVA SDK 定制开发数据传输服务

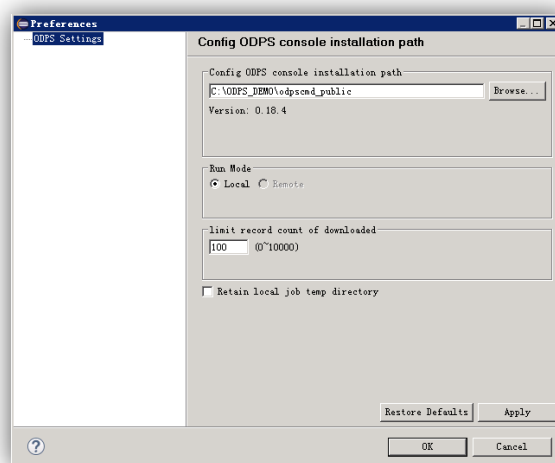
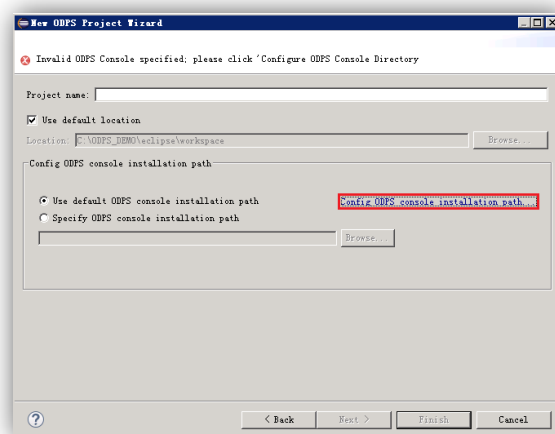
2.6.1 实验： 安装配置 eclipse 开发环境：

- (1) 将 C:\ODPS_DEMO\InstallMedia\odps-eclipse-plugin-bundle-0.16.0.jar 拷贝至目录 C:\ODPS_DEMO\eclipse\plugins\
- (2) 执行 C:\ODPS_DEMO\eclipse\eclipse.exe, 打开 eclipse, 点击 New -> Other

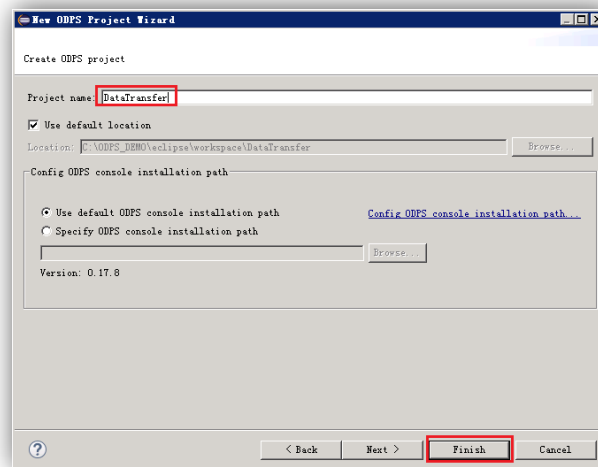




(3) 配置 odps



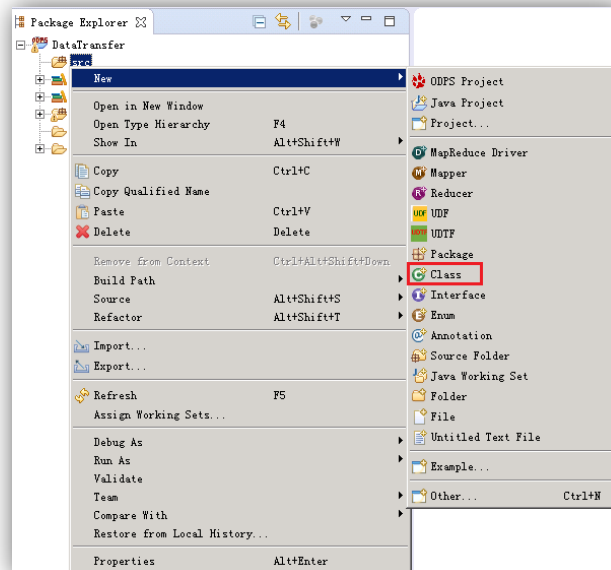
(4) 指定 Java Project Name:



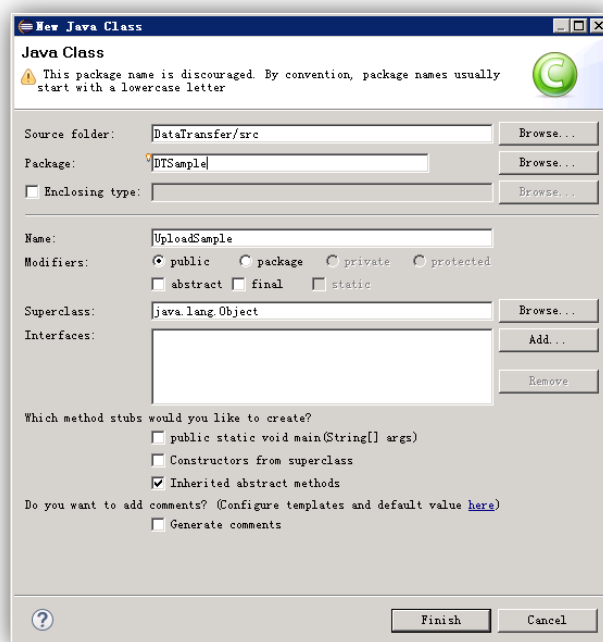
2.6.2 实验：单线程上传文件：

本实验可参考脚本：C:\ODPS_DEMO\resources\02-DataTransfer\UploadSample.java

(1) 新增 Java 类：



(2) 类名为 UploadSample，包名为 DTSample



(3) 设计该类实现功能为将单个文件上传至 ODPS 的表中，需要的输入参数为：

```
-f <file_name>
-t <table_name>
-c <config_file>
[-p <partition spec>]
[-fd <field_delimiter>]
```

编写方法 printUsage，提示调用语法：

```
//// 打印调用语法
private static void printUsage(String msg) {
    System.out.println(
        "Usage: uploadSample -f file \\n"
        + "          -t table\\n"
        + "          -c config_file \\n"
        + "          [-p partition] \\n"
        + "          [-fd field_delimiter] \\n"
    );
    if (msg != null) {
        System.out.println(msg);
    }
}
```

编写获取、解析输入参数的方法：

```
//// 获得、解析输入参数
private static String accessId;
private static String accessKey;
private static String odpsEndpoint;
private static String TunnelEndpoint;
private static String project;
private static String table;
private static String partition;
private static String fieldDelimiter;
private static String file;
private static void parseArgument(String[] args) {
    for (int i = 0; i < args.length; i++) {
        if ("-f".equals(args[i])) {
            if (++i == args.length) {
                throw new IllegalArgumentException("source file not specified in -f");
            }
            file = args[i];
        }
        else if ("-t".equals(args[i])) {
            if (++i == args.length) {
                throw new IllegalArgumentException("ODPS table not specified in -t");
            }
            table = args[i];
        }
        else if ("-c".equals(args[i])) {
            if (++i == args.length) {
                throw new IllegalArgumentException(
                    "ODPS configuration file not specified in -c");
            }
            try {
                InputStream is = new FileInputStream(args[i]);
                Properties props = new Properties();
                props.load(is);
                accessId = props.getProperty("access_id");
                accessKey = props.getProperty("access_key");
                project = props.getProperty("project_name");
                odpsEndpoint = props.getProperty("end_point");
                TunnelEndpoint = props.getProperty("tunnel_endpoint");
            } catch (IOException e) {
                throw new IllegalArgumentException(
                    "Error reading ODPS config file '" + args[i] + "'.");
            }
        }
    }
}
```



```
else if ("-p".equals(args[i])){
    if (++i == args.length) {
        throw new IllegalArgumentException(
            "odps table partition not specified in -p");
    }
    partition = args[i];
}
else if ("-fd".equals(args[i])){
    if (++i == args.length) {
        throw new IllegalArgumentException(
            "odps table partition not specified in -p");
    }
    fieldDelimiter = args[i];
}
}
if(file == null) {
    throw new IllegalArgumentException(
        "Missing argument -f file");
}
if (table == null) {
    throw new IllegalArgumentException(
        "Missing argument -t table");
}

if (accessId == null || accessKey == null ||
    project == null || OdpsEndpoint == null || TunnelEndpoint == null) {
    throw new IllegalArgumentException(
        "ODPS conf not set, please check -c odps.conf");
}
}
```

(4) 编写方法，从文件中读出记录，同时将这些记录格式化：

```
//// 读出记录，逐列处理
import java.text.DecimalFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import com.aliyun.odps.Column;
import com.aliyun.odps.OdpsType;
import com.aliyun.odps.TableSchema;
```

```
import com.aliyun.odps.data.ArrayRecord;
import com.aliyun.odps.data.Record;
class RecordConverter {
    private TableSchema schema;
    private String nullTag;
    private SimpleDateFormat dateFormatter;

    private DecimalFormat doubleFormat;
    private String DEFAULT_DATE_FORMAT_PATTERN = "yyyyMMddHHmmss";
    public RecordConverter(TableSchema schema, String nullTag,
        String dateFormat, String tz) {
        this.schema = schema;
        this.nullTag = nullTag;
        if (dateFormat == null) {
            this.dateFormatter = new SimpleDateFormat(DEFAULT_DATE_FORMAT_PATTERN);
        } else {
            dateFormatter = new SimpleDateFormat(dateFormat);
        }
        dateFormatter.setLenient(false);
        dateFormatter.setTimeZone(TimeZone.getTimeZone(tz == null ? "GMT" : tz));

        doubleFormat = new DecimalFormat();
        doubleFormat.setMinimumFractionDigits(0);
        doubleFormat.setMaximumFractionDigits(20);
    }
    /**
     * record to String array
     */
    public String[] format(Record r) {
        int cols = schema.getColumns().size();
        String[] line = new String[cols];
        String colValue = null;
        for (int i = 0; i < cols; i++) {
            Column column = schema.getColumn(i);
            OdsType t = column.getType();
            switch (t) {
                case BIGINT: {
                    Long v = r.getBigint(i);
                    colValue = v == null ? null : v.toString();
                    break;
                }
                case DOUBLE: {
                    Double v = r.getDouble(i);
```

```
        if (v == null) {
            colValue = null;
        } else {
            colValue = doubleFormat.format(v).replaceAll(",", "");
        }
        break;
    }
    case DATETIME: {
        Date v = r.getDatetime(i);
        if (v == null) {
            colValue = null;
        } else {
            colValue = dateFormatter.format(v);
        }
        break;
    }
    case BOOLEAN: {
        Boolean v = r.getBoolean(i);
        colValue = v == null ? null : v.toString();
        break;
    }
    case STRING: {
        String v = r.getString(i);
        colValue = (v == null ? null : v.toString());
        break;
    }
    default:
        throw new RuntimeException("Unknown column type: " + t);
}

if (colValue == null) {
    line[i] = nullTag;
} else {
    line[i] = colValue;
}
}
return line;
}

/**
 * String array to record
 * @throws ParseException
 */
public Record parse(String[] line){
```

```
if (line == null) {
    return null;
}

int columnCnt = schema.getColumns().size();
Column[] cols = new Column[columnCnt];
for (int i = 0; i < columnCnt; ++i) {
    Column c = new Column(schema.getColumn(i).getName(),
        schema.getColumn(i).getType());
    cols[i] = c;
}
ArrayRecord r = new ArrayRecord(cols);
int i = 0;
for (String v : line) {
    if (v.equals(nullTag)) {
        i++;
        continue;
    }
    if (i >= columnCnt) {
        break;
    }
    OdsType type = schema.getColumn(i).getType();
    switch (type) {
        case BIGINT:
            r.setBigint(i, Long.valueOf(v));
            break;
        case DOUBLE:
            r.setDouble(i, Double.valueOf(v));
            break;
        case DATETIME:
            try {
                r.setDatetime(i, dateFormatter.parse(v));
            } catch (ParseException e) {
                throw new RuntimeException(e.getMessage());
            }
            break;
        case BOOLEAN:
            v = v.trim().toLowerCase();
            if (v.equals("true") || v.equals("false")) {
                r.setBoolean(i, v.equals("true") ? true : false);
            } else if (v.equals("0") || v.equals("1")) {
                r.setBoolean(i, v.equals("1") ? true : false);
            } else {
```

```

        throw new RuntimeException(
            "Invalid boolean type, expect: true|false|0|1");
    }
    break;
case STRING:
    r.setString(i, v);
    break;
default:
    throw new RuntimeException("Unknown column type");
}
i++;
}
return r;
}
}

```

(5) 编写主方法，读取文件，上传到 odps 表中去：

```

//// 主方法
public static void main(String args[]) {
    try {
        parseArgument(args);
    } catch (IllegalArgumentException e) {
        printUsage(e.getMessage());
        System.exit(2);
    }

    Account account = new AliyunAccount(accessId, accessKey);
    Odps odps = new Odps(account);
    odps.setDefaultProject(project);
    odps.setEndpoint(OdpsEndpoint);
    BufferedReader br = null;

    try {
        TableTunnel tunnel = new TableTunnel(odps);
        tunnel.setEndpoint(TunnelEndpoint);
    }
}

```

```
TableTunnel.UploadSession uploadSession = null;

if(partition != null) {
    PartitionSpec spec = new PartitionSpec(partition);
    System.out.println(spec.toString());
    uploadSession=tunnel.createUploadSession(project, table,spec);
    System.out.println("Session Status is : " +
        uploadSession.getStatus().toString());
} else {
    uploadSession= tunnel.createUploadSession(project, table);
}

Long blockid = (long) 0;
RecordWriter recordwriter = uploadSession.openRecordWriter(blockid,
    true);
Record record = uploadSession.newRecord();
TableSchema schema = uploadSession.getSchema();
RecordConverter converter = new RecordConverter(schema, "NULL", null,
    null);

br = new BufferedReader(new FileReader(file));
Pattern pattern = Pattern.compile(fieldDelimiter);
String line = null;
while ((line = br.readLine()) != null) {
    String[] items=pattern.split(line,0);
    record = converter.parse(items);
    recordwriter.write(record);
}

recordwriter.close();

Long[] blocks = {blockid};
uploadSession.commit(blocks);

System.out.println("Upload succeed!");
```

```

    } catch (TunnelException e) {

        e.printStackTrace();

    } catch (IOException e) {

        e.printStackTrace();

    }finally {

        try {

            if (br != null)

                br.close();

        } catch (IOException ex) {

            ex.printStackTrace();

        }

    }

}

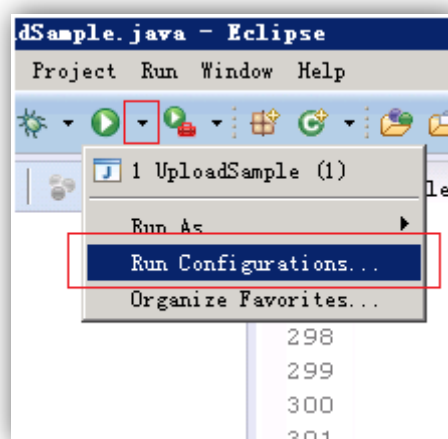
}

```

(6) 在 ODPS 中建表:

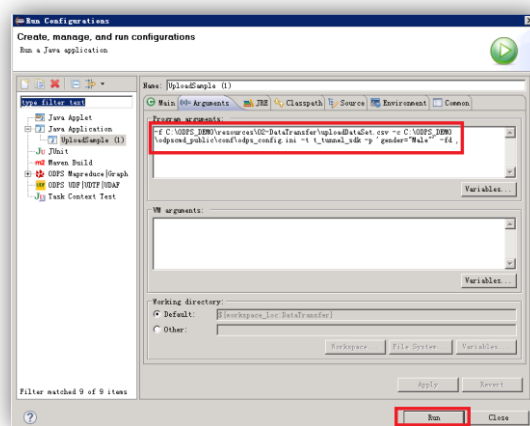
```
odpscmd -f C:\ODPS_DEMO\resources\02-DataTransfer\crt_tbl.sql
```

(7) 在 eclipse 中设置测试运行参数:

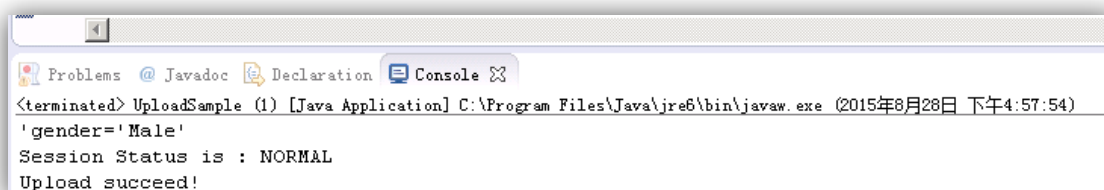


将下列参数填入 program parameter:

```
-f C:\ODPS_DEMO\resources\02-DataTransfer\uploadDataSet.csv -t t_tunnel_sdk
-p 'gender="Male"' -fd , -c C:\ODPS_DEMO\odpscmd_public\conf\odps_config.ini
```



通过 console 查看输出信息：



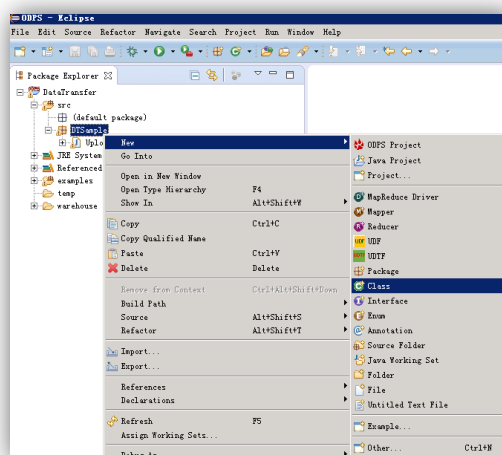
去 ODPS project 里检查上传结果：

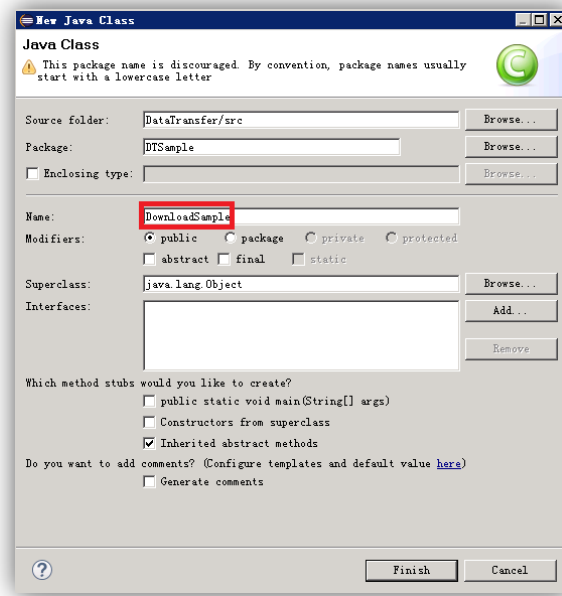
```
read t_tunnel_sdk;
```

2.6.3 实验：单线程下载文件：

本实验可参考脚本：C:\ODPS_DEMO\resources\02-DataTransfer\DownloadSample.java

(1) 在已有的名称为 DataTransfer 的 Java project 中的 DTSample 包下新增 Java 类：





- (2) 编写方法 `printUsage`，提示调用该程序的输入参数；
- (3) 编写方法 `parseArgument`，获取并解析输入参数；
- (4) 编写类 `RecordConverter` 用来格式化数据，生成 `Record` 记录；
- (5) 编写方法 `main` 方法，实现单线程数据下载：

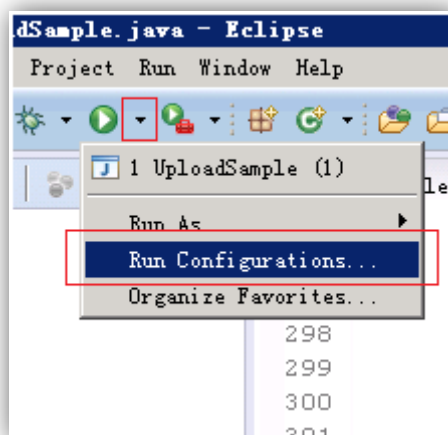
```
//根据输入参数中的配置文件，配置阿里云账号
Account account = new AliyunAccount(accessId, accessKey);
Odps odps = new Odps(account);
odps.setDefaultProject(project);
odps.setEndpoint(OdpsEndpoint);
```

```
//基于上述云账号，创建服务入口类
TableTunnel tunnel = new TableTunnel(odps);
tunnel.setEndpoint(TunnelEndpoint);
```

```
//创建从上述 odps 服务通道中下载数据的会话，分为分区的表和非分区表两种：
TableTunnel.DownloadSession session;
if(partition != null) {
    PartitionSpec spec = new PartitionSpec(partition);
    session= tunnel.createDownloadSession(project, table, spec);
}else{
    session= tunnel.createDownloadSession(project, table);
}
```

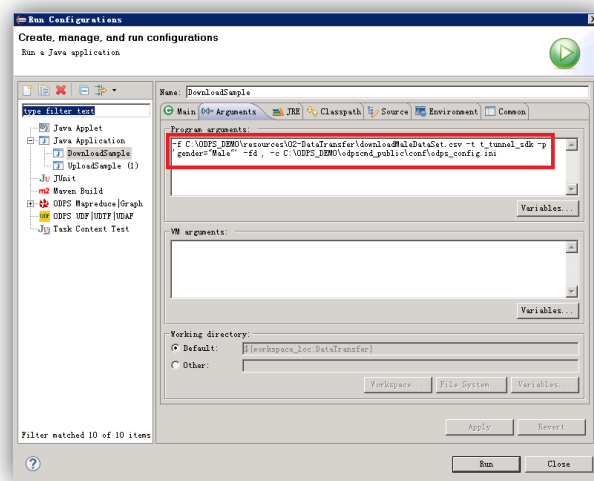
```
//从 odps 表中读出记录，格式化后，写入到本地文件：
RecordReader reader = session.openRecordReader(OL, session.getRecordCount(),
    true);
TableSchema schema = session.getSchema();
Record record;
RecordConverter converter = new RecordConverter(schema, "NULL", null, null);
String[] items = new String[schema.getColumns().size()];
while ((record = reader.read()) != null) {
    items = converter.format(record);
    for(int i=0; i<items.length; ++i) {
        if(i>0) out.write(fieldDelimiter.getBytes());
        out.write(items[i].getBytes());
    }
    out.write(lineDelimiter.getBytes());
}
reader.close();
out.close();
```

(6) 在 eclipse 中设置测试运行参数：

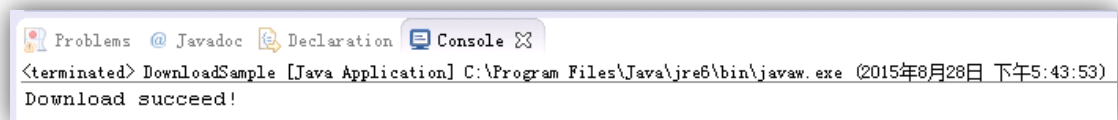


将下列参数填入 program parameter:

```
-f C:\ODPS_DEMO\resources\02-DataTransfer\downloadMaleDataSet.csv -fd , -c
C:\ODPS_DEMO\odpscmd_public\conf\odps_config.ini -t t_tunnel_sdk -p
'gender="Male"'
```



通过 console 查看输出信息：



去查看下载文件 C:\ODPS_DEMO\resources\02-DataTransfer\downloadMaleDataSet.csv，
并和 ODPS 表 t_tunnel_sdk 中的数据对比。

2.7 ODPS SQL 运算符

2.7.1 实验：建立测试用表，并加载数据：

- (1) 切换至目录 C:\ODPS_DEMO\resources\03-SQL
- (2) 执行建表语句：

```
odpscmd -f t_sign.crt.sql
```

- (3) 进入 odpscmd 交互界面，加载数据 t_sign.csv：

```
tunnel upload t_sign.csv t_sign;
```

2.7.2 实验：使用关系型运算符：

(1) 常见的比较类运算符：

```
// 等号、不等、大于、小于、is null、is not null、in、like 等
select * from t_sign where id = 11111;
select * from t_sign where name <> 'Danny';
select * from t_sign where height > 180;
select * from t_sign where is_female = true;
select * from t_sign where birth_day >= '1980-01-01 00:00:00';
select * from t_sign where is_female is null;
select * from t_sign where birth_day is not null;
select * from t_sign where id in (11111,33333);
select * from t_sign where name like 'D%';
select * from t_sign where name like '____';
```

(2) rlike 运算符：

```
//找出 name 全是小写字母的记录
select * from t_sign where name rlike '^[a-z]+$';
//找出 name 里包含空格的记录
select * from t_sign where name rlike ' +';
//找出 name 里全是数字的记录
select * from t_sign where name rlike '^[0-9]+$';
//找出 name 中包含数字的记录
select * from t_sign where name rlike '^.+[0-9].+$';
```

(3) 使用运算符时包含 NULL 值或者空串：

```
select t1.* from t_sign t1 join t_sign t2 on t1.id = t2.id;
select /*+mapjoin(t1)*/ t1.*
  from t_sign t1 join t_sign t2
    on t1.birth_day > t2.birth_day;
select * from t_sign where is_female in (true,false);
select * from t_sign where height not in (178.4,180.2,171.8);
//产生值中包含空串的记录
insert into table t_sign select 1, '',2.0,true,'1900-01-01 00:00:00' from dual;
//检查 not in 的时候，对 NULL 值和空串的不同匹配结果
select * from t_sign where name not in ('Adam','Danny','ella');
```

2.7.3 实验：使用算术运算符：

(1) 常见的算术运算符：

```
select id+height, height-id, height%10.2, -height, height*height,height/10.2
from t_sign;
```

(2) double 类型运算时注意精度：

```
select * from t_sign where height*0.618 = 106.1724;
select * from t_sign where abs(height*0.618- 106.1724) < 0.00001;
```

2.7.4 实验：使用位运算符：

```
select conv('11111',10,2),conv('22222',10,2),
       11111|22222, 11111&22222
from dual;
```

2.7.5 实验：使用逻辑运算符：

(1) 常见的逻辑运算符：

```
select * from t_sign where id<=33333 and birth_day > '1980-01-01 00:00:00';
select * from t_sign where id<=33333 or birth_day > '1980-01-01 00:00:00';
select * from t_sign
where not(id<=33333) and birth_day > '1980-01-01 00:00:00';
```

(2) 逻辑运算符中的项包含 NULL 值：

```
select id<0, height>180, id<0 and height > 180 from t_sign;
select id<0, height>180, id<0 or height > 180 from t_sign;
select height>180, not(height > 180) from t_sign;
```

2.7.6 实验：类型的显示转换：

Boolean 类型不能转换成任何其他类型，任何类型也不能转换成 Boolean 类型。Datetime 可以转换成 Sting，符合格式的 string 也可以转换成 Datetime。类型 bigint,double 和 string(符合格式) 可以互转。

```
select cast(id as double) as i2d,  
       cast(id as string) as i2s,  
       cast(height as bigint) as d2i,  
       cast(height as string) as d2s,  
       cast(birth_day as string) as dt2s,  
       cast('12345' as bigint) as s2i,  
       cast('1.2345' as double) as s2d,  
       cast('2015-10-01 00:00:00' as datetime) as s2dt  
from t_sign;
```

2.7.7 实验：类型的隐式转换：

```
select id+height+'123',  
       concat(name,id,height,birth_day)  
from t_sign;
```

2.8 ODPS DDL 基本操作

切换到目录 C:\ODPS_DEMO\resources\03-SQL，执行 odpscmd，进入交互界面。

2.8.1 实验：使用 SQL 建表：

(1) 创建不带分区的表：

```
drop table if exists t_ddl;  
create table t_ddl(  
    id bigint comment "Identified Number",  
    name string comment "Name of Person")  
comment "test table for DDL"  
lifecycle 1;
```

(2) 创建带分区的表:

```
drop table if exists t_ddl_p;
create table t_ddl_p(
  id bigint comment "Identified Number",
  name string comment "Name of Person")
comment "test partitioned table for DDL"
partitioned by (gender string comment "Gender of Person")
lifecycle 1;
```

(3) CTAS 建表:

```
// 为已建成的表添加测试数据
insert into table t_ddl select * from t_tunnel;
// 测试 create as
create table t_ddl_as as select * from t_ddl;
desc t_ddl_as;
read t_ddl_as;
// 测试 create like
create table t_ddl_like like t_ddl;
desc t_ddl_like;
read t_ddl_like;
// 测试分区表的 create as
create table t_ddl_p_as as select * from t_ddl_p;
desc t_ddl_p_as;
// 测试分区表的 create like
create table t_ddl_p_like like t_ddl_p;
desc t_ddl_p_like;
```

2.8.2 实验：分区操作：

(1) 添加分区:

```
alter table t_ddl_p add if not exists partition (gender='Male');
```

(2) 删除分区:

```
alter table t_ddl_p drop if exists partition (gender='Male');
```

2.8.3 实验：修改表属性：

(1) 添加列：

```
alter table t_ddl add columns (age bigint comment "Age of Person");  
alter table t_ddl_p add columns (age bigint comment "Age of Person");
```

(2) 修改列名：

```
alter table t_ddl change column age rename to age_new;  
alter table t_ddl_p change column age rename to age_new;
```

(3) 修改生命周期：

```
alter table t_ddl set lifecycle 10;  
alter table t_ddl_p set lifecycle 10;
```

(4) 禁止/启用分区表分区的生命周期：

```
alter table t_ddl_p partition(gender='Male') disable lifecycle;
```

(5) 修改表注释：

```
alter table t_ddl set comment "test table for DDL--Modified";
```

(6) 修改表的修改时间：

```
alter table t_ddl touch;  
alter table t_ddl_p touch;  
alter table t_ddl_p touch partition (gender='Male');
```

(7) 修改分区值：

```
alter table t_ddl_p partition (gender='Male') rename to partition(gender='M');
```

(8) 修改表名：

```
alter table t_ddl_p rename to t_ddl_pp;
```


2.8.4 实验：视图操作：

(1) 创建视图：

```
create or replace view v_t_ddl (id comment "Identified Number")
  comment "view from t_ddl"
as select id from t_ddl;
alter table t_ddl_pp rename to t_ddl_p;
create view if not exists v_t_ddl_p (id) as select id from t_ddl_p;
```

(2) 修改视图名称：

```
alter view v_t_ddl_p rename to v_t_ddl_pp;
```

(3) 删除视图：

```
drop view v_t_ddl_pp;
drop view v_t_ddl;
```

2.9 ODPS DML 基本操作

2.9.1 实验：建表并准备数据：

(1) 切换至目录 C:\ODPS_DEMO\resources\03-SQL

(2) 执行建表语句：

```
odpscmd -f dml.crt.sql
```

(3) 进入 odpscmd 交互界面，加载数据 t_dml.csv：

```
tunnel upload t_dml.csv t_dml;
```

2.9.2 实验：查询操作

(1) 常见的简单查询：

```
// 行过滤和列选择
select * from t_dml where province='浙江省';
select city, amt from t_dml where sale_date >='2015-05-23 00:00:00';
select distinct city from t_dml where amt > 700;
```

(2) 使用子句的查询:

```
// group by + order by + limit
select city,sum(amt) as total_amt
  from t_dml
 where province='浙江省'
 group by city
 having count(*)>1
       and sum(amt) > 2000
 order by total_amt desc
 limit 10;
// distribute by + sort by
select city, cnt, amt
  from t_dml
distribute by city
 order by cnt;
```

2.9.3 实验：更新数据：

(1) 追加插入记录：

```
// insert into table: 追加插入
insert into table t_dml select -1,'1900-01-01 00:00:00',' ',' ',0,0,0 from dual;
select * from t_dml where id=-1;
alter table t_dml_p add if not exists partition (sale_date='2015-01-01');
insert into table t_dml_p partition (sale_date='2015-01-01')
select -1, ' ', ' ', 0, 0, 0 from dual;
select * from t_dml_p where sale_date='2015-01-01';
```

(2) 覆盖插入记录：

```
// 覆盖原有记录，插入新纪录
insert overwrite table t_dml
select -2,'1900-01-01 00:00:00', ' ', ' ',0,0,0 from dual;
select * from t_dml where detail_id in (-1,-2);
```

```
// 分区表覆盖分区数据
alter table t_dml_p add if not exists partition (sale_date='2015-01-01');
insert overwrite table t_dml_p partition (sale_date='2015-01-01')
select -2, '', '', 0, 0, 0 from dual;
select * from t_dml_p;
// 使用覆盖插入的方式清空非分区表
insert overwrite table t_dml select * from t_dml where 1=2;
count t_dml;
// 使用覆盖插入的方式清空某个分区
insert overwrite table t_dml_p partition(sale_date='2015-01-01')
select detail_id, province, city, product_id, cnt, amt from t_dml_p where 1=2;
select count(*) from t_dml_p where sale_date='2015-01-01';
// 也可以通过删除分区的方式清空分区
alter table t_dml_p drop if exists partition (sale_date='2015-01-01');
select count(*) from t_dml_p where sale_date='2015-01-01';
// 恢复测试数据
tunnel upload t_dml.csv t_dml;
```

2.9.4 实验：多路输出：

将表 t_dml 中的数据 detail_id 编号大于 5340000 的插入一张临时表备查，并将 5 月 1 日的记录插入分区表 t_dml_p 的 20150501 分区中去，将 5 月 2 日的数据插入 20150502 分区：

```
alter table t_dml_p add if not exists partition (sale_date='20150501');
alter table t_dml_p add if not exists partition (sale_date='20150502');
create table t_dml_01 like t_dml;
from t_dml
insert into table t_dml_01
select detail_id,sale_date,province,city,product_id,cnt,amt
where detail_id > 5340000
insert overwrite table t_dml_p partition (sale_date='20150501')
select detail_id,province,city,product_id,cnt,amt
where sale_date >= '2015-05-01 00:00:00'
and sale_date <= '2015-05-01 23:59:59'
insert overwrite table t_dml_p partition (sale_date='20150502')
select detail_id,province,city,product_id,cnt,amt
where sale_date >= '2015-05-02 00:00:00'
and sale_date <= '2015-05-02 23:59:59'
;
```

2.9.5 实验：动态分区：

将表 t_dml 中的数据插入到分区表 t_dml_p 中去，由于分区个数多，不能手工逐个处理，太麻烦，需要动态分区：

```
alter table t_dml_p drop if exists partition (sale_date='20150501');
alter table t_dml_p drop if exists partition (sale_date='20150502');
insert into table t_dml_p partition(sale_date)
select detail_id, province, city, product_id, cnt, amt,
       to_char(sale_date, 'yyyymmdd') as sale_date
  from t_dml;
insert overwrite table t_dml_p partition(sale_date)
select detail_id, province, city, product_id, cnt, amt,
       to_char(sale_date, 'yyyymmdd') as sale_date
  from t_dml;
```

2.9.6 实验：join 以及 mapjoin HINT：

(1) 切换至目录 C:\ODPS_DEMO\resources\03-SQL

(2) 执行建表语句：

```
odpscmd -f t_product.crt.sql
```

(3) 进入 odpscmd 交互界面，加载数据 t_product.csv：

```
tunnel upload t_product.csv t_product;
```

(4) 普通 join:

事实表 t_dml 包含了销售记录信息，其中字段 product_id 为产品标识，可以关联另一张维表 t_product 获得产品的说明信息，现在想通过 SQL 得到针对产品大类的销售金额统计：

```
//按照产品分类（category_name）统计销售金额
//1-left outer join
select t2.category_name, sum(t1.amt)
  from t_dml t1
 left outer join t_product t2
    on t1.product_id=t2.product_id
 group by t2.category_name;
//2-inner join (join)
select t2.category_name, sum(t1.amt)
  from t_dml t1
 inner join t_product t2
    on t1.product_id=t2.product_id
 group by t2.category_name;
//3-right outer join
select t2.category_name, sum(t1.amt)
  from t_dml t1
 right outer join t_product t2
    on t1.product_id=t2.product_id
 group by t2.category_name;
select t1.category_name, sum(t2.amt)
  from t_product t1
 right outer join t_dml t2
    on t1.product_id=t2.product_id
 group by t1.category_name;
//4-full outer join
select t2.category_name, sum(t1.amt)
  from t_dml t1
 full outer join t_product t2
    on t1.product_id=t2.product_id
 group by t2.category_name;
```

在本需求中，关联条件都是等值关联，可以用普通的 join（各种 join）去实现。

(5) mapjoin HINT:

由于各种原因，造成销售信息表 t_dml 中的记录存在一些质量问题，可能的问题包括：

- 1- 产品标识错误： 可以通过单价判断，单价相等的标识不同，则可能存在错误
- 2- 价格错误： 如果销售记录中的平均单价高于产品维表中的定价，则可能存在问题

请协助发现这些可能存在问题的记录。

```
// left outer join 实现质量问题 1:
select /*+mapjoin(t2)*/t1.*,t1.amt/t1.cnt,t2.product_id,t2.price
  from t_dml t1
 left outer join t_product t2
    on t1.product_id<>t2.product_id
    and t1.amt/t1.cnt = t2.price
 where t2.price is not null;
//2-inner join (join) 实现质量问题 2:
select /*+mapjoin(t1)*/ t1.*, t1.amt/t1.cnt,t2.product_id, t2.price
  from t_dml t1
 inner join t_product t2
    on t1.product_id=t2.product_id
    or t1.amt/t1.cnt - t2.price < 0.01;
//3-right outer join: 重写 left outer join 实现的逻辑
select /*+mapjoin(t1)*/t2.*,t2.amt/t2.cnt,t1.product_id,t1.price
  from t_product t1
 right outer join t_dml t2
    on t1.product_id<>t2.product_id
    and t2.amt/t2.cnt = t1.price
 where t1.price is not null;
```

在做关联时，如果关联条件比较复杂（比如包含 or 等连接条件）或者是关联条件中存在非等值关联（比如大于、小于或者不等于等），则普通的 join 无法实现，可以采用带有 mapjoin HINT 的 join 方式。

2.9.7 实验：子查询：

ODPS SQL 支持将子查询作为一张表来用，可以用于简单查询、join 等。在使用中，必须为子查询指定别名。

```
// 子查询用于简单查询:
select * from (select distinct province from t_dml) t;
// 子查询用于 join:
select t2.category_name, sum(t1.amt)
  from (select * from t_dml where amt > 800) t1
 inner join t_product t2
    on t1.product_id=t2.product_id
 group by t2.category_name;
//子查询用于 mapjoin
select /*+mapjoin(t1)*/t2.*,t2.amt/t2.cnt,t1.product_id,t1.price
  from t_product t1
 right outer join (select * from t_dml where detail_id > 5340000) t2
    on t1.product_id<>t2.product_id
    and t2.amt/t2.cnt = t1.price
 where t1.price is not null;
```

2.9.8 实验: UNION ALL:

在销售记录中, 由于实际售卖价钱和产品的标称价并不一致, 我想获得产品的所有出现过的单价 (包括实际售卖价和标称价)

```
select * from (
  select product_id,price, 'STD' type from t_product
  UNION ALL
  select distinct product_id, amt/cnt as price, 'USED' type from t_dml
) t
order by product_id,type,price desc
limit 100;
```

2.9.9 实验: CASE WHEN 表达式:

市场部准备做一次市场营销活动, 对于一次购买 3-5 个产品的, 在目前的售价上实行 9 折优惠, 一次购买 6 个及以上产品的, 给与 8 折优惠。请基于 5 月份数据想评估一下此次活动的成本 (为了简单可行, 活动成本定义为目前销售额减掉优惠后的销售额)。

```
select sum(amt)-sum(case when cnt>=6 then amt*0.8
                        when cnt>=3 then amt*0.9
                        else amt
                      end) cost,
from t_dml;
```

2.10 内置函数

2.10.1 实验：数值类函数：

(1) 三角函数类：

已知三角形两边长度为 4, 5，夹角为 30 度，求三角形面积

```
select 0.5*3*4*cos(30/180*3.1415926) from dual;
```

(2) 数字整形类：

对数字进行加工处理，请分别显示数字 3.1415926 的向上取整值、向下取整值、四舍五入保留 3 位小数的值、截掉小数位的值以及用二进制来表示该值。

```
select ceil(3.1415926),floor(3.1415926),round(3.1415926,3),
       trunc(3.1415926),conv('3.1415926',10,2)
from dual;
```

(3) 随机函数类：

```
select rand() from dual;
select rand(detail_id),rand() from t_dml limit 10;
```

(4) 综合使用

使用蒙特卡洛法求 π 值的近似值：产生一系列的成对的随机数，根据每队随机数到点 (0.5, 0.5) 的距离可判断该点是否在单位圆内，计算落在圆内的点占所有点的比例，即可得到 π 值的近似值：


```
// 产生约 10 万对随机点进行近似值计算：
select (inCircle/totalCnt)/pow(0.5,2) as PI
  from (select count(*) as totalCnt,
            sum(case when sqrt(pow((x-0.5),2)+pow((y-0.5),2)) <0.5 then 1
                    else 0
                end) inCircle
        from (select /*+mapjoin(t2)*/ rand() as x,rand() as y
              from (select * from t_dml limit 10000) t1
              left outer join (select * from t_dml limit 10) t2
              on t1.detail_id <> t2.detail_id) tt
        ) t;

// 产生约 100 万对随机点进行近似值计算：
select (inCircle/totalCnt)/pow(0.5,2) as PI
  from (select count(*) as totalCnt,
            sum(case when sqrt(pow((x-0.5),2)+pow((y-0.5),2)) <0.5 then 1
                    else 0
                end) inCircle
        from (select /*+mapjoin(t2)*/ rand() as x,rand() as y
              from (select * from t_dml limit 10000) t1
              left outer join (select * from t_dml limit 100) t2
              on t1.detail_id <> t2.detail_id) tt
        ) t;
```

2.10.2 实验：字符串类函数：

(1) 长度类：

```
select province,length(province),lengthb(province) from t_dml limit 10;
```

(2) 查找类：

目前销售记录中，哪些省、市名字比较接近？

```
select province, city, char_matchcount(province, city) as sim
  from (select distinct province, city
        from t_dml) t
 order by sim desc
 limit 10;
```

目前销售记录中，省份的第一个字在城市名中是否出现？有没有出现多次的？

```
select province, city,
       instr(city,substr(province,1,3),1,1) as FirstPos,
       case when instr(city,substr(province,1,3),1,2) = 0 then 'No'
             else 'Yes'
       end as SecondPos
from (select distinct province, city
      from t_dml) t
order by SecondPos desc, FirstPos desc
limit 10;
```

(3) 转换类：

要把数据从一个编码为 utf8 的库导入到一个字符集为 gb2312 的库中，其中有些繁体字，如“阿裏雲”等字样，请问会出现乱码的情况吗？

```
select is_encoding('阿裏雲', 'utf-8', 'gb2312') from dual;
```

(4) 整形类：

```
select concat(province, '|',city) from t_dml limit 10;
select category_name, tolower(split_part(category_name,' ',2))
from t_product;
```

2.10.3 实验：日期类函数：

(1) 日期获取：

```
//根据日期，截取部分信息
select dt,
       datepart(dt, 'yyyy') as year,
       datepart(dt, 'mm') as month,
       datepart(dt, 'dd') as day,
       datepart(dt, 'hh') as hour,
       datepart(dt, 'mi') as minute,
       datepart(dt, 'ss') as second
from (select getdate() dt from dual) t;
```

```
// 日期截取
select datetrunc('2015-01-31 02:30:45', 'dd') from dual;
// 获得具体日期
select getdate(),lastday(getdate()),
       weekday(getdate()),weekofyear(getdate())
from dual;
```

(2) 日期转换:

```
//字符串转成日期, 日期转换成字符串
select to_date('20150131','yyyymmdd'),
       to_char('2015-01-31 00:00:00', '日期: yyyymmdd')
from dual;
// Unix 时间和 ODPS 时间互转
select from_unixtime(1), unix_timestamp('2015-10-01 00:00:00') from dual;
// 判断字符串是否满足预定义的日期格式
select sale_date, isdate(sale_date, 'yyyymmdd') from t_dml_p limit 10;
```

(3) 日期运算:

统计 5 月 1 日从产品 5 第一次成交后一小时三十分钟内 (含), 产品 5 销量 (含第一次成交) 占同期总销量的比例:

```
select /*+mapjoin(t2)*/
       sum(case when product_id=5 then cnt else 0 end)/sum(cnt)
from t_dml t1
join (select min(sale_date) as begin_dt,
            dateadd(dateadd(min(sale_date),1,'hh'),30, 'mi') as end_dt
      from t_dml
      where product_id=5
            and datetrunc(sale_date,'dd')='2015-05-01 00:00:00')t2
on t1.sale_date >= t2.begin_dt
and t1.sale_date <= t2.end_dt;
```

日期相减:

```
select max(sale_date), min(sale_date)
       datediff(max(sale_date),min(sale_date),'dd')
from t_dml;
```

2.10.4 实验：窗口函数：

(1) 统计量类：

根据 5 月份销售数据，统计出日销量波动最小的产品（即变标准差最小）。

```
select product_id,stddev(amt)over(partition by product_id) std_dev
  from (select datetrunc(sale_date,'dd') as dt, product_id, sum(amt) as amt
        from t_dm1
        group by datetrunc(sale_date,'dd'), product_id) t1
 order by std_dev
 limit 1;
```

(2) 排名类：

根据 5 月份销售数据，统计出同一产品成交最短时间间隔（以产品 1 为例，列出出两次成交时间差最小的记录）。

```
select datediff(t1.sale_date, t2.sale_date, 'ss') dt_diff,
       t1.detail_id, t2.detail_id
  from (select row_number()over(partition by product_id order by sale_date)
        as id, *
        from t_dm1
        where product_id=1
        and datetrunc(sale_date,'mm')='2015-05-01 00:00:00') t1
 join (select row_number()over(partition by product_id order by sale_date)
        as id, *
        from t_dm1
        where product_id=1
        and datetrunc(sale_date,'mm')='2015-05-01 00:00:00') t2
    on t1.id = t2.id+1
 order by dt_diff
 limit 1;
```

(3) 带 rows 的开窗：

在做时序分析(Time Series Analysis)时，对于长期趋势(Trend)的分析最常见的是使用移动平均法(Moving Average method)，是通过逐期移动时间序列，并计算一系列扩大时间间隔后的序时平均数，最终形成一个新时间序列的方法。优点是由其它因素而引起的变动影响

被削弱了，对原序列起到了修匀的作用，从而更清晰地呈现出现象的变动趋势。通常 MA 是由专业的挖掘算法来实现，我们可以尝试使用带 rows 的开窗函数来实现：

以 4 天作为一个平滑窗口的宽度（前 2 后 1），即取 $n-2$ 天到 $n+1$ 天作为一个平滑窗口，计算连续四天内的均值作为第 n 天的代表值。对产品 1 和产品 2 的销售金额和销售量进行平稳化。做趋势图，分别做横向（不同产品的趋势图）和纵向（同一产品平滑前后的趋势图）比较。

```
select dt, product_id,
       avg(amt) over(partition by product_id order by dt rows between 2
                     preceding and 1 following),
       avg(cnt) over(partition by product_id order by dt rows between 2
                     preceding and 1 following)
from (select to_char(sale_date,'yyyymmdd') as dt, product_id,
            sum(cnt) as cnt, sum(amt) as amt
      from t_dml where product_id in (1,2)
      group by to_char(sale_date,'yyyymmdd'),product_id) t;
```

2.10.5 实验：聚合函数：

(1) 统计量类：

给出销售信息表 t_dml 中的不同产品的销售金额的基本统计信息。

```
select product_id, count(*), min(amt), max(amt), sum(amt), avg(amt),
       median(amt), stddev(amt), stddev_samp(amt)
from t_dml
group by product_id;
```

(2) 字符串类：

将产品标称单价在 50-100 元的，生成一个清单，不同产品名称之间用|分隔开。

```
select wm_concat('|',product_name)
from t_product
where price >=50
and price<=100;
```

2.10.6 实验：其他函数：

(1) COALESCE 处理 NULL 值：

将 t_sign 中的名字(name)和生日(birth_day)拼成一个串

```
select concat(coalesce(name,'unknown'),coalesce(birth_day,'unknown'))
from t_sign;
```

(2) decode 分支函数

将销售记录 t_dml 中浙江、上海和北京的销量单独统计出来：

```
select decode(province,'浙江省','浙','上海市','沪','北京市','京','其他'),
       sum(cnt)
from t_dml
group by decode(province,'浙江省','浙','上海市','沪','北京市','京','其他');
```

(3) ordinal 有序位置函数：

顺序统计量（Ordered Statistics，也称次序统计量）是非参统计的重要组成部分，适用于整体分布不能由有限个参数表示的情况。利用 ordinal 函数，可以方便的计算顺序统计相关的一些基础统计量：

最小顺序统计量 $X_{(1)}$ ，最大顺序统计量 $X_{(n)}$ ，极差 $R=X_{(n)} - X_{(1)}$ ，四分卫极差：

$IQL= X_{(0.75*n)} - X_{(0.25*n)}$

```
select ordinal(8,1,2,3,4,6,5,8,7,9,0)- ordinal(3,1,2,3,4,6,5,8,7,9,0) as IQL
from dual;
```

(4) sample 采样函数：

通过采样分析的手段，从销售记录表 t_dml 中得到 1/100 的数据，分析采样样本，试着推断总体的销售金额的平均值、标准差、极值、极差等，然后从总体中计算出这些统计量进行验证。调整采样比例，重复上述推断过程，找到一个准确程度和样本体量的平衡点，进一步思考：这个平衡点有多大参考价值？

```
// 1/100 比例的样本的推断值
select avg(amt) as Average,stddev_samp(amt) as Standard_Dev,
       min(amt) as Min_Val, max(amt) as Max_Val,
       max(amt) - min(amt) as Range_Val,count(*) as Sample_size
  from (select amt from t_dml where sample(100,1,detail_id)=true) t;
// 1/10 比例的样本的推断值
select avg(amt) as Average,stddev_samp(amt) as Standard_Dev,
       min(amt) as Min_Val, max(amt) as Max_Val,
       max(amt) - min(amt) as Range_Val,count(*) as Sample_size
  from (select amt from t_dml where sample(10)=true) t;
//总体实际值
select avg(amt) as Average,stddev_samp(amt) as Standard_Dev,
       min(amt) as Min_Val, max(amt) as Max_Val,
       max(amt) - min(amt) as Range_Val,count(*) as Sample_size
  from t_dml;
```

2.11 执行计划和 Logviewer

(1) 使用 explain 查看执行计划，和课件中的执行计划作比对：

```
explain insert into table t_test1
select a.product_id, sum(b.price)
  from t_dml a
 join t_product b
   on a.product_id=b.product_id
 group by a.product_id;
```

(2) 执行 sql，根据返回的 logviewer 查看图形化的执行信息：

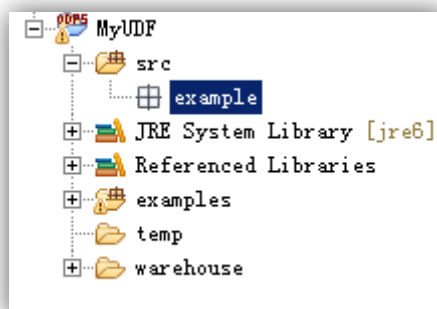
```
insert into table t_test1
select a.product_id, sum(b.price)
  from t_dml a
 join t_product b
   on a.product_id=b.product_id
 group by a.product_id;
```

2.12 用户自定义函数 UDF

2.12.1 实验：搭建配置实验环境：

确认已搭建实验环境。如果尚未搭建，请参照实验 2.6.1 的步骤，完成实验环境配置。

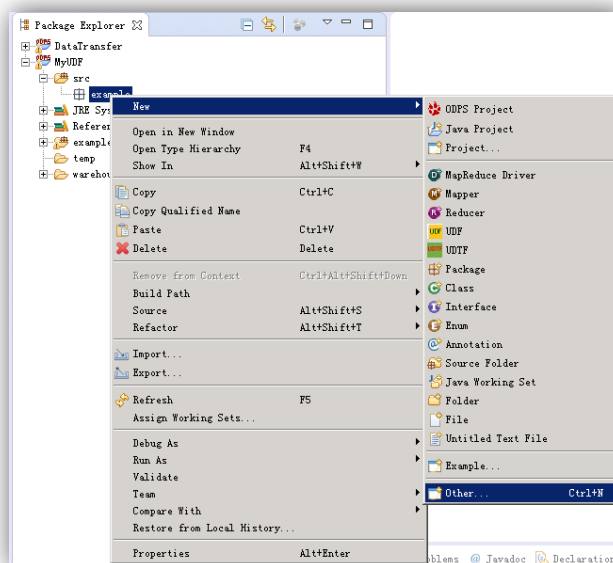
新建 odps project，命名为 MyUDF，在该 project 下生成名称为 example 的 package。

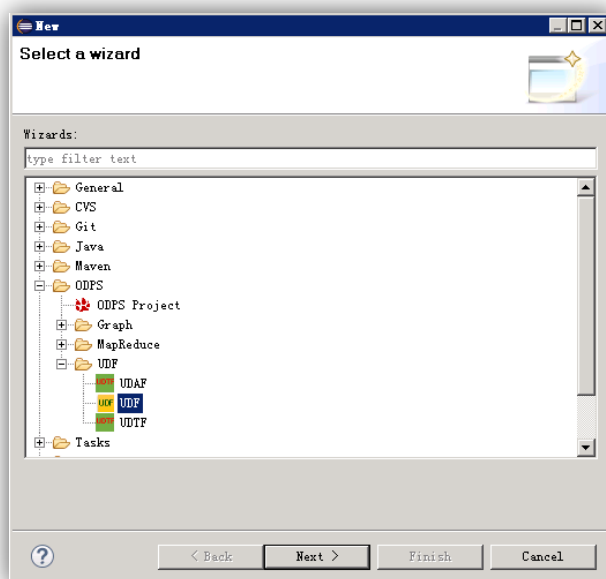


2.12.2 实验：最简单的 UDF 示例 LowCase

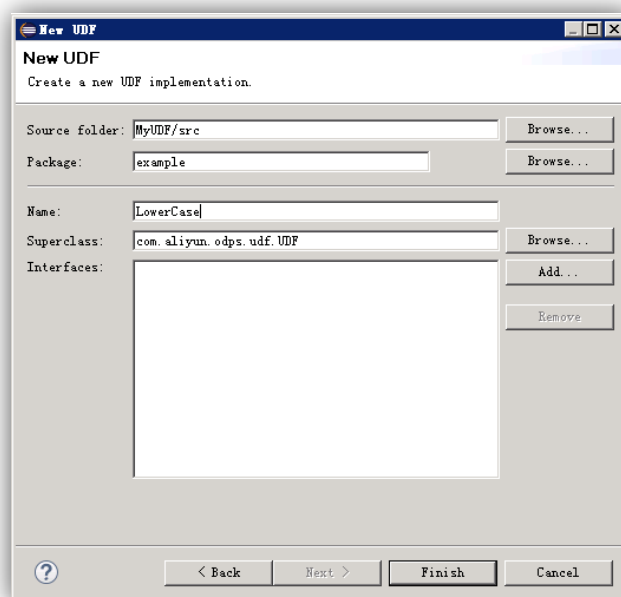
本实验可参考脚本：C:\ODPS_DEMO\resources\04-UDF\LowerCase.java

(1) 新增 Java 类：





(2) 类名为 LowerCase:



(3) 编写 java 代码:

在生成的代码框架中，补充处理逻辑。

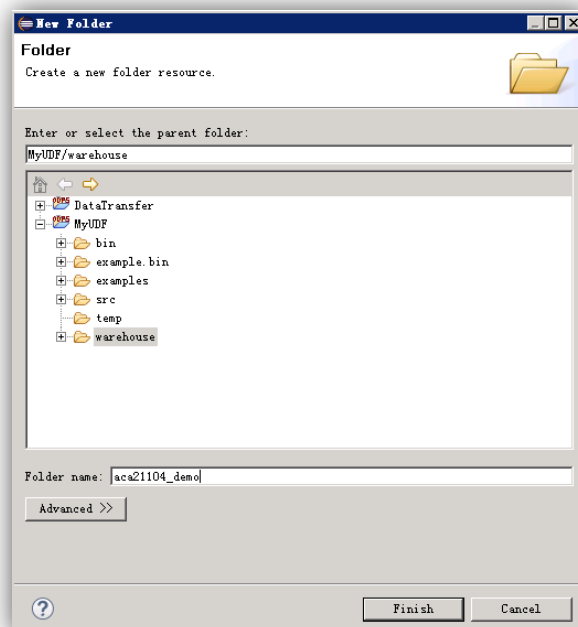
我们希望 LowerCase 能把输入的字符串中的字母转换成小写字母，对于其他字符不作处理。LowerCase 类继承了 com.aliyun.odps.udf.UDF 类，需要重写方法 evaluate，把我们的处理逻辑增加到该方法中。

```
// 重写 evaluate 方法，把大写转成小写，注意 null 的处理
public String evaluate(String s) {
    if (s == null) {
        return null;
    }
    return s.toLowerCase();
}
```

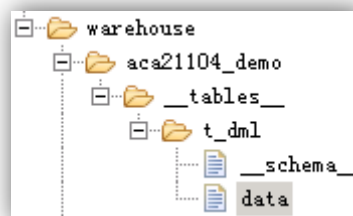
(4) 测试 java 代码（本地模式）：

1-准备测试数据：

在当前 project MyUDF 下找到目录 warehouse, 点击右键，依次选择 new -> folder:



将目录命名为：aca21104_demo, 同样的操作依次建立如下图所示的目录和文件列表：

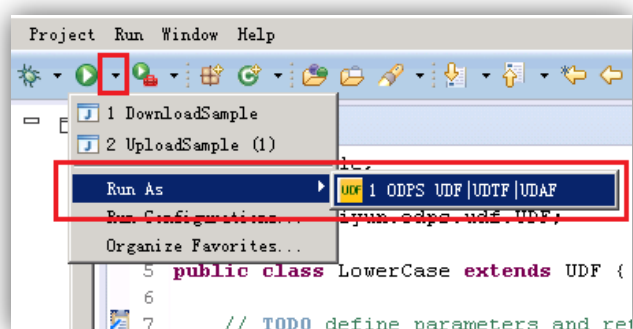


双击 __schema__, 在文件中增加如下内容：

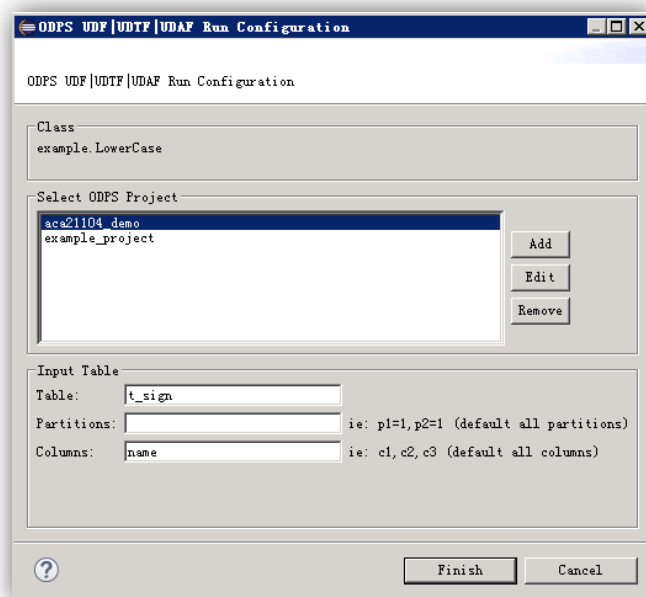
```
project=aca21104_demo
table=t_dml
columns=detail_id:bigint,sale_date:datetime,province:string,
city:string,product_id:bigint,cnt:bigint,amt:double
```

双击 data,在文件中增加和 ODPS 中的 t_dml 中一致的数据。按照同样的步骤,在 warehouse 目录的下的项目 aca21104_demo 中增加表 t_sign, t_dml_p, 两者和 odps 项目 aca21104_demo 中同名的表结构一致, 数据为后者的子集。同时在 ODPS 和本地添加一张表 t_udtf, 包含两个字段: name:string, score:double, 其中, name 的值中, 包含空格分隔的多个人名, 类似于"Tom Jerry Delubii", 创建完成后的显示结构图为:

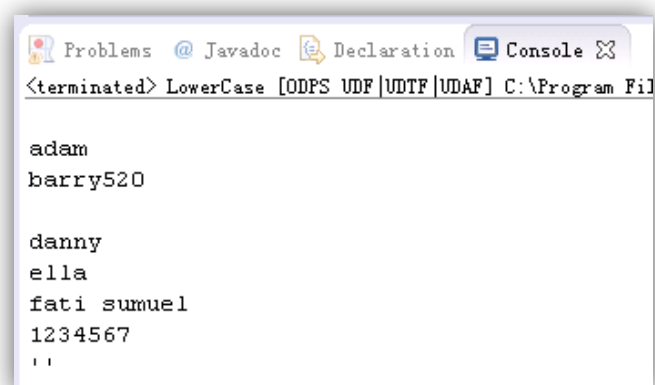
2-测试: 选择 Run -> Run As -> ODPS UDF|UDTF|UDAF :



填写配置信息:

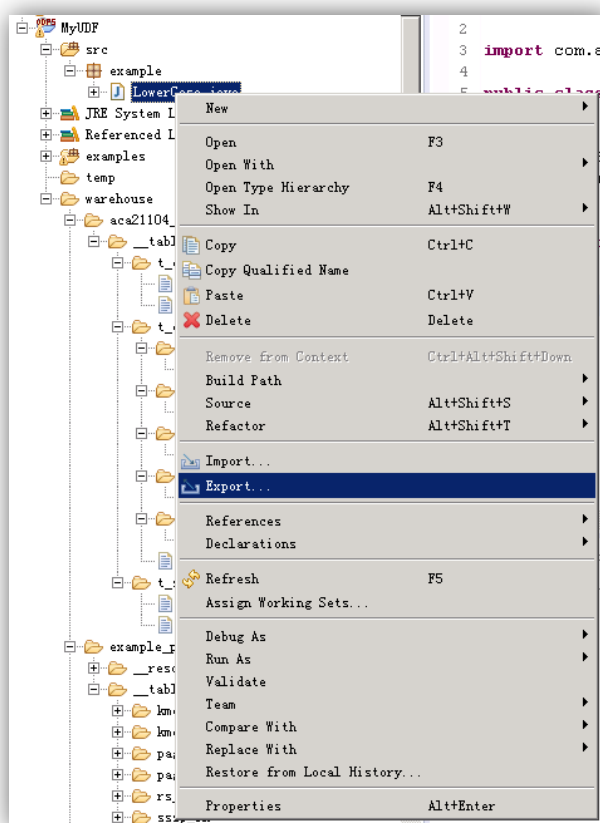


运行，并查看结果：

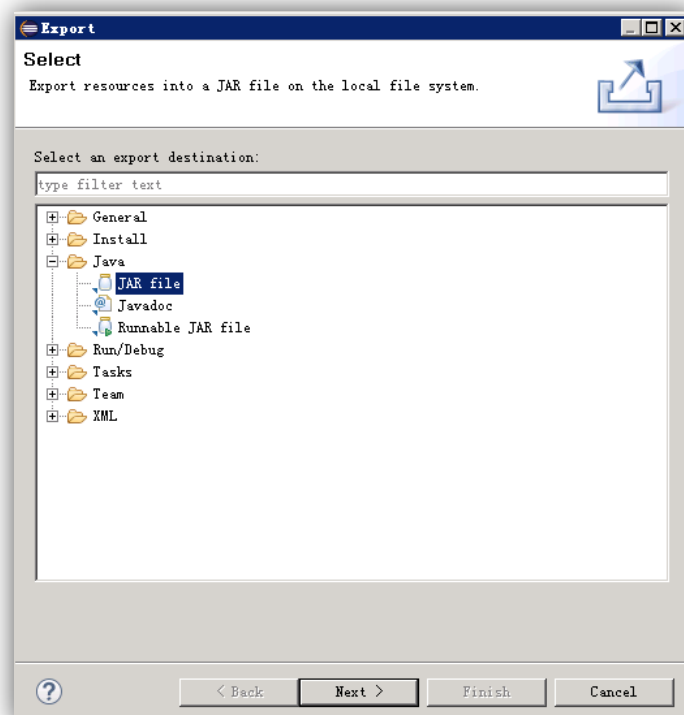


(5) 导出 jar 包：

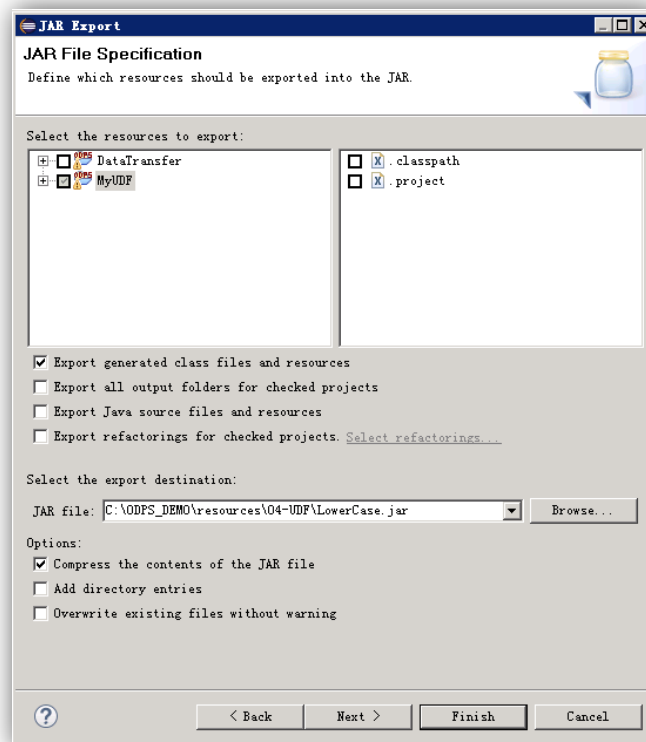
右键点击 LowerCase.java, 在弹出菜单中选择 Export:



选择导出类型为 Jar file:



选择导出目录，指定文件名字，完成导出。



(6) 将 Jar 包添加到 ODPS:

在命令行中目录切换到 C:\ODPS_DEMO\resources\04-UDF，执行 odpscmd 进入 ODPS 交互操作界面。

将 jar 包添加到 odps 的项目 aca21104_demo 中去:

```
// 使用 add 命令添加
add jar LowerCase.jar;
// 如果要覆盖已存在资源的话，需要使用 -f 选项
add jar LowerCase.jar -f;
```

基于 UDF 的 jar 包创建 function，并测试 function:

```
create function lowercase as example.LowerCase using LowerCase.jar;
select name, lowercase(name) from t_sign;
```

2.12.3 实验：UDF 之最简单的加密

本实验可参考脚本：C:\ODPS_DEMO\resources\04-UDF\easyEncryption.java

- (1) 在项目 MyUDF 下的包 example 下新增 UDF class，名字为 easyEncryption;
- (2) 编写 java 代码：重写方法 evaluate，完成逻辑处理：

```
// 重写 evaluate 方法,
public String evaluate(String s) {
    if (s == null) { return null; }
    char ss[]=s.toCharArray();
    char[] res = s.toCharArray();
    for(int i=0; i<s.length(); i++) {
        if (Character.isLowerCase(ss[i])) {
            if (ss[i]=='z'){
                res[i]='A';
            } else {
                res[i]=(char)((int)Character.toUpperCase(ss[i]) + 1);
            }
            //System.out.println(res[i]);
        } else if (Character.isUpperCase(ss[i])) {
            if (ss[i]=='A'){
                res[i]='z';
            } else {
                res[i]=(char)((int)Character.toLowerCase(ss[i]) - 1);
            }
        }
    }
    return new String(res);
}
```

- (3) 本地测试后到出 Jar 包，添加到 ODPS 的项目 aca21104_demo 中，并创建 function，测试、应用：

```
add jar easyEncryption.jar;
create function encrypt as example.easyEncryption using easyEncryption.jar;
select name,encrypt(name) from t_sign;
```

2.12.4 实验：UDTF 之多值列拆分：散伙分行李

本实验可参考脚本：C:\ODPS_DEMO\resources\04-UDF\byeFellows.java

(1) 在项目 MyUDF 下的包 example 下新增 UDTF class，名字为 byeFellows;

(3) 编写 java 代码：重写方法 setup(),process(),close()，完成逻辑处理：

```
// 重写 process 方法,
public void process(Object[] args) throws UDFException {
    String a = (String) args[0];
    Long b = (Long) args[1];
    int cnt=1;
    char[] c = a.toCharArray();
    for (int i=0; i<a.length(); i++){
        if (c[i] == ' '){
            cnt=cnt+1;
        }
    }
    if (cnt==0) cnt=1;
    for (String t: a.split("\\s+")) {
        forward(t, (double)b/cnt);
    }
}
```

(3) 本地测试后到出 Jar 包，添加到 ODPS 的项目 aca21104_demo 中，并创建 function，测试、应用：

```
add jar byeFellows.jar;
create function splitMultiValue
as example.byeFellows using byeFellows.jar;
create table t_udtf(name string, score bigint);
tunnel upload t_udtf.csv t_udtf;
select splitMultiValue(name,score) as (name,score) from t_udtf;
```

2.12.5 实验：UDAF 之变异系数

本实验可参考脚本：C:\ODPS_DEMO\resources\04-UDF\calcCV.java

(1) 在项目 MyUDF 下的包 example 下新增 UDAF class，名字为 calcCV;

(2) 编写 java 代码: 重写方法 `iterate()`, `terminate()`, `merge()`, `newBuffer()`:

```
// 重写 newBuffer 方法,
private static class AvgBuffer implements Writable {
    private double sum = 0;
    private long count = 0;
    private double sum2 = 0;
    public void write(DataOutput out) throws IOException {
        out.writeDouble(sum);
        out.writeDouble(sum2);
        out.writeLong(count);
    }
    public void readFields(DataInput in) throws IOException {
        sum = in.readDouble();
        sum2 = in.readDouble();
        count = in.readLong();
    }
}
public Writable newBuffer() {
    return new AvgBuffer();
}
```

```
// 重写 iterate 方法,
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();
        buf.sum2 += Math.pow(arg.get(), 2);
    }
}
```

```
// 重写 merge 方法,
public void merge(Writable buffer, Writable partial) throws UDFException {
    AvgBuffer buf = (AvgBuffer) buffer;
    AvgBuffer p = (AvgBuffer) partial;
    buf.sum += p.sum;
    buf.sum2 += p.sum2;
    buf.count += p.count;
}
```

```
// 重写 terminate 方法,
public writable terminate(Writable buffer) throws UDFException {
    AvgBuffer buf = (AvgBuffer) buffer;
    if (buf.count == 0) {
        ret.set(0);
    } else if (buf.count == 1){
        ret.set(buf.sum);
    } else {
        ret.set(Math.sqrt((buf.sum2-buf.sum*(buf.sum/buf.count))/(buf.count-1
        ))/(buf.sum/buf.count));
    }
    return ret;
}
```

(3) 本地测试后到出 Jar 包, 添加到 ODPS 的项目 aca21104_demo 中, 并创建 function, 测试、应用:

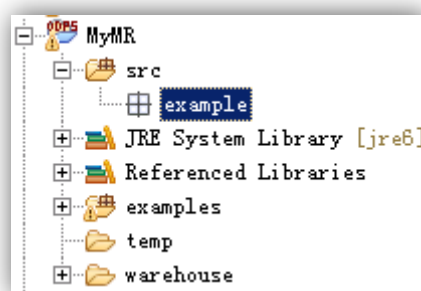
```
add jar calcCV.jar;
create function cv as example.calcCV using calcCV.jar;
select cv(amt),product_id from t_dm1
group by product_id;
//其他方法计算变异系数, 验证结果
select stddev_samp(amt)/avg(amt),product_id from t_dm1
group by product_id;
```

2.13 MapReduce

2.13.1 实验: 搭建配置实验环境:

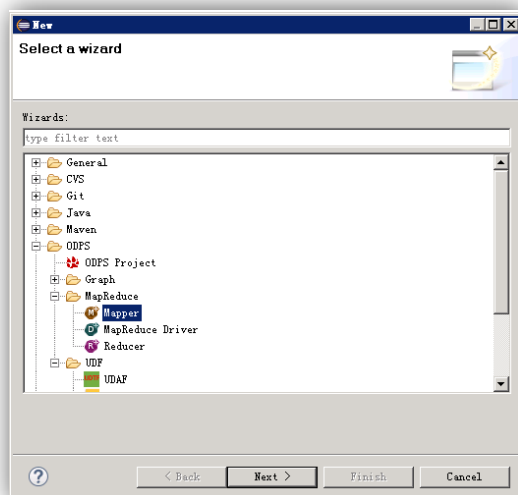
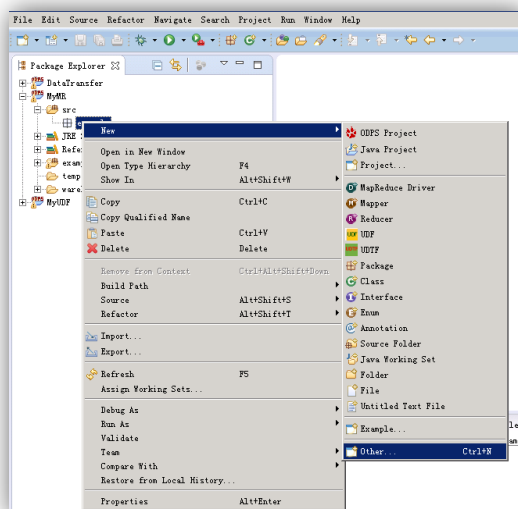
确认已搭建实验环境。如果尚未搭建, 请参照实验 2.6.1 的步骤, 完成实验环境配置。

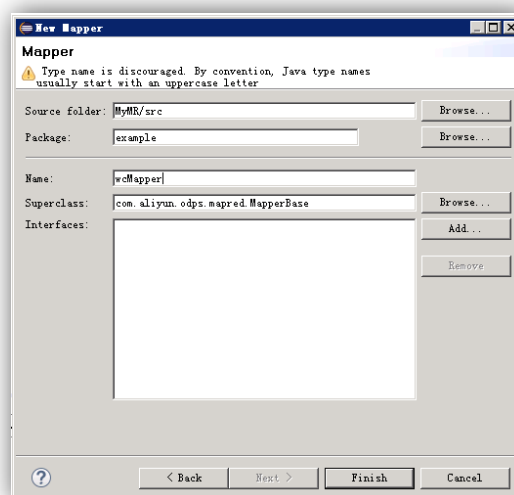
新建 odps project, 命名为 MyMR, 在该 project 下生成名称为 example 的 package。



2.13.2 实验：Word Count, MR 界的 “Hello World! ”

(1) 新增 Mapper 类，命名为 wcMapper:





- (2) 同步骤(1)，新增 Reducer 类，命名为 wcReducer
- (3) 同步骤(1)，新增 Mapreduce Driver 类，命名为 wcDriver
- (4) 完成 Mapper 的处理逻辑（可参考 C:\ODPS_DEMO\resources\05-MR\wcMapper.java），主要是重写 setup 方法，初始化上下文变量，创建输出记录；重写 map 方法，将输入记录逐条处理，将出现过的词装入词集，并标注数量 1：

//重写 setup()方法，初始化

```
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());  
}
```

//重写 map()方法，逐条处理记录

```
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);  
    }  
}
```

(5) 完成 Reducer 的处理逻辑(可参考 C:\ODPS_DEMO\resources\05-MR\wcReducer.java):

主要是重写 setup 方法, 初始化上下文变量, 创建输出记录; 重写 reduce 方法, 将从 map 输出的记录(排序后的, key-value 对儿)逐条处理, 处理方式为将相同 key (map 输出的 key 值)的记录做成一个迭代器, 逐个 key 值去累加出现的次数 (map 输出的 value 值):

//重写 setup()方法, 初始化

```
public void setup(TaskContext context) throws IOException {  
    result = context.createOutputRecord();  
}
```

//重写 reduce()方法, 逐条处理记录

```
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);  
}
```

(6) 完成 Mapreduce Driver 的处理逻辑:

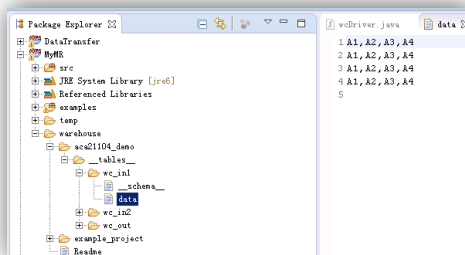
(可参考 C:\ODPS_DEMO\resources\05-MR\wcDriver.java):

主要是去提供一些和 Job 相关的信息, 包括:

- 1- 指定 Map 的输出信息, 包括键信息 (key)、键值 (Value) 等
- 2- 指定 MR 的输入和输出表
- 3- 指定 Mapper 和 Reducer, 如果有 combiner 的话, 也需要指定 (本例中没有使用 Combiner)
- 4- 需要的话, 可以指定 Job 的配置参数 (本例使用缺省值)
- 5- 提交、运行 Job
- 6- 等待程序运行完成

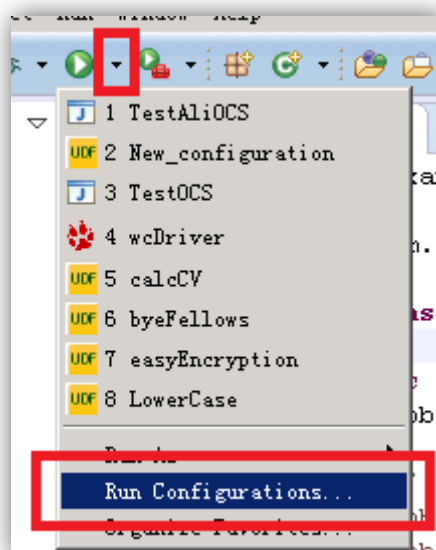
```
//重新 main () 函数，提供各种注册信息，并提交运行 job
// TODO: specify map output types
job.setMapOutputKeySchema(SchemaUtils.fromString("word:string"));
job.setMapOutputValueSchema(SchemaUtils.fromString("count:bigint"));
// TODO: specify input and output tables
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(),job);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(),job);
// TODO: specify a mapper
job.setMapperClass(WCMapper.class);
// TODO: specify a reducer
job.setReducerClass(WCReducer.class);
RunningJob rj = JobClient.runJob(job);
rj.waitForCompletion();
```

(7) 准备测试数据，可参考 2.12.2 的第 4 步，wc_in1 中数据如下：



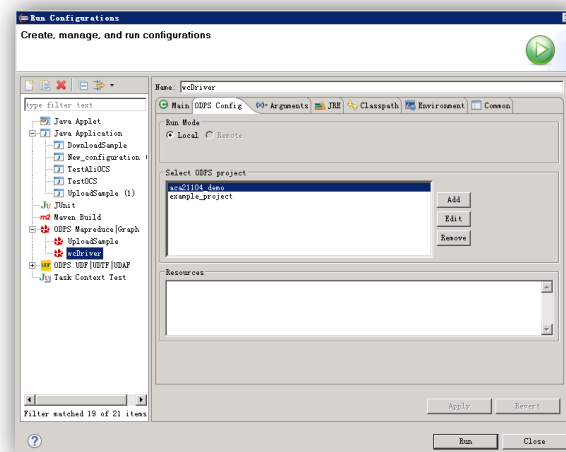
(8) 使用本地模式测试：

首先对 wcDriver.java 的运行参数进行配置：

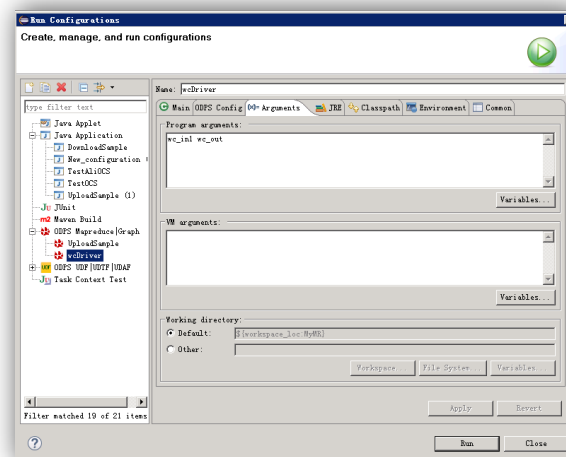


选中测试使用的本地项目：

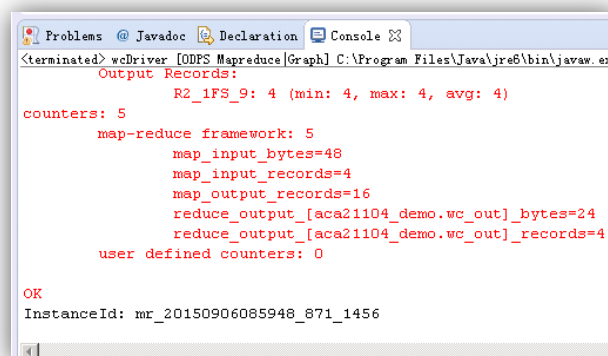
深入阿里云产品



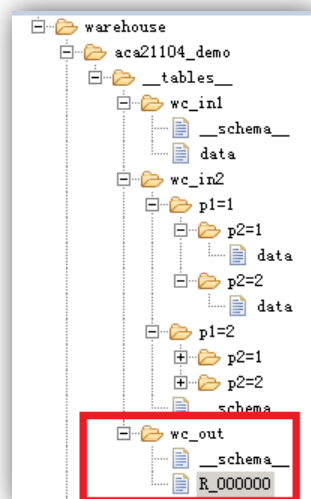
填入输入参数：



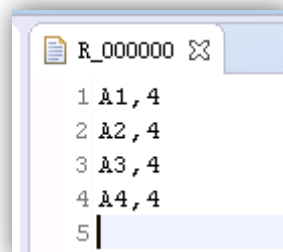
执行：



执行成功之后，打开 warehouse 下的项目 aca21104_demo 下的表 wc_out，刷新目录，找到对应的数据文件：



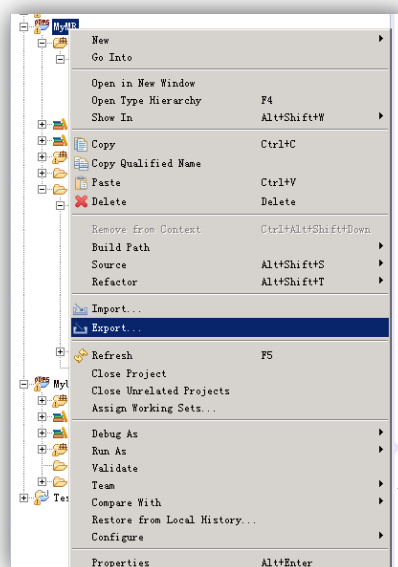
检查 wc_out 下的数据文件：



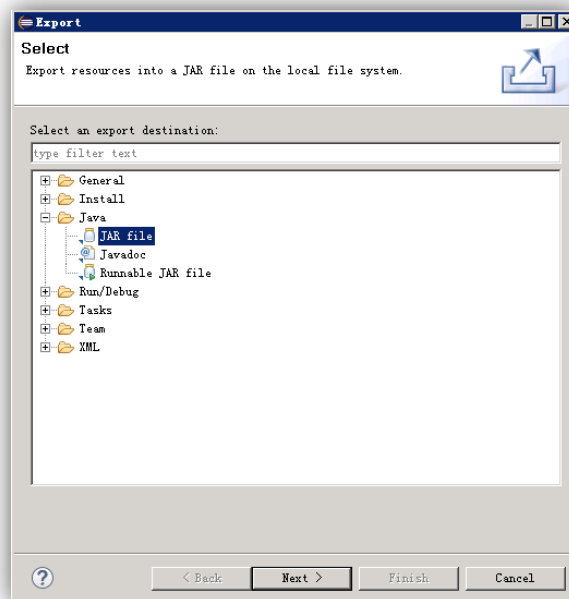
输出结果符合预期，测试成功完成。

(9) 将 MR 程序生成 jar 包：

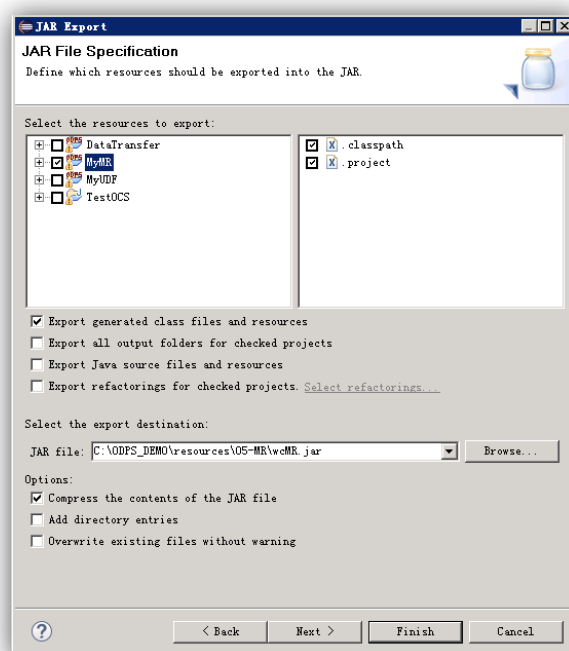
在项目上点右键，选择 export：



选择导出类型为 jar:



指定导出路径和文件名:



(10) 将 jar 包作为资源上传到 ODPS 的 aca21104_demo 中去:

打开 odps 客户端，进入交互模式:

```
cd C:\ODPS_DEMO\resources\05-MR\
odpscmd
```

通过命令 add jar 将 jar 上传至 ODPS:

```
add jar C:\ODPS_DEMO\resources\05-MR\wcMR.jar -f;
```

注意: -f 参数为覆盖该项目中已存在的同名的资源:

(11) 在 ODPS 中测试、使用 MR

准备测试数据:

```
drop table wc_in;
drop table wc_out;
drop table constitution;
drop table law_freq;
create table wc_in (col1 string, col2 string, col3 string,col4 string);
tunnel upload wc_in.csv aca21104_demo.wc_in;
create table constitution (word string);
tunnel upload constitution.csv aca21104_demo.constitution;
create table wc_out(word string, cnt bigint);
create table law_freq(word string, cnt bigint);
```

使用 jar 命令, 调用 jar 包:

```
jar -resources wcMR.jar -classpath wcMR.jar example.wcDriver wc_in wc_out;
read wc_out;
```

```
jar -resources wcMR.jar -classpath wcMR.jar example.wcDriver
    constitution law_freq;
read wc_out;
```

2.13.3 实验: 好友推荐

- (1) 继续使用 Java Project: myMR
- (2) 新增一个 ODPS Mapper 类, 名字为 FriendMapper
- (3) 新增一个 ODPS Reducer 类, 名字为 FriendCombiner

- (4) 新增一个 ODPS Reducer 类，名字为 FriendReducer
- (5) 新增一个 ODPS Driver 类，名字为 FriendDriver
- (6) 完成 Mapper 的处理逻辑（参考 C:\ODPS_DEMO\resources\05-MR\FriendMapper.java）
主要是重写 setup 方法，初始化上下文变量，创建输出记录；重写 map 方法，将输入记录逐条处理，将出现过的词装入词集，并标注数量 1：

//重写 setup()方法，初始化

```
public void setup(TaskContext context) throws IOException {
    key = context.createMapOutputKeyRecord();
    value = context.createMapOutputValueRecord();
}
```

//重写 map()方法，逐条处理记录

```
.....
String user = record.get(0).toString();
String all = user + " " + record.get(1).toString();
String[] arr = all.split(" ");
Arrays.sort(arr);
int len = arr.length;
for (int i=0; i<len-1; ++i) {
    for(int j=i+1; j<len; ++j) {
        key.set(new Object[] {arr[i] + " " + arr[j]});
        if(arr[i].equals(user)) {
            value.set(new Object[] {0L});
            context.write(key, value);
        }
        else {
            context.write(key,value);
            value.set(new Object[] {1L});
        }
    }
}
.....
```

- (7) 完成 Combiner 的处理（可参考 C:\ODPS_DEMO\resources\05-MR\FriendCombiner.java）

关于 Combiner 的使用，需要根据实际情况判断是否有必要。一般来讲，Mapper 的结果如果在相同的分片中同一个 Key 值对应的重复记录数比较多，可以考虑使用 combiner，减少

传递给 reducer 的数据量，节省网络带宽，提高运行效率。本例中，可根据测试数据（ODPS 表 FRIENDS_LIST）进行粗略的判断。

Combiner 中的处理逻辑非常简单，针对 Mapper 的输出结果，在分片范围内，对 key 值进行汇总处理（本例中为计数累加）：即在同一个 Mapper 输出结果的分片内，对 key 相同的记录进行记录数累加统计。

初始化时重新定义 Combiner 的输出，Combiner 的输出将取代 Mapper 的输出，被传到 Reducer 中去处理：

```
//重写 setup()方法，初始化
public void setup(TaskContext context) throws IOException {
    result=context.createMapOutputValueRecord();
}
```

Combiner 的逻辑处理在 reducer 方法中实现：对每一个 Mapper 分片输出的 Key 值进行汇总：

```
//重写 reduce()方法，逐条处理记录
public void reduce(Record key, Iterator<Record> values, TaskContext context)
throws IOException {
    long count = 0;
    while (values.hasNext()) {
        Record val = values.next();
        if( 0 == (Long)val.get(0) ) {
            count = 0;
            break;
        }
        count += (Long)val.get(0);
    }
    result.set(new Object[] { count });
    context.write(key, result);
}
```

(8) 完成 Reducer 的处理（可参考 C:\ODPS_DEMO\resources\05-MR\FriendReducer.java）

事实上 Reducer 的处理逻辑和 Combiner 中的逻辑是一样的，即将输入到 Reducer 的记录按照 Key 统计，得到计数结果，然后基于计数结果再去实现其他的业务逻辑。

但是有一个非常明显的区别，提交到 Combiner 的数据，同一个 key 值的记录可能分散在不同的分片里，但是提交到 Reducer 的数据，都是经过 shuffle 之后的，即同一个 key 值对应的记录，只会提交给相同的 reducer。

//重写 setup()方法，初始化

```
public void setup(TaskContext context) throws IOException {  
    result=context.createMapOutputValueRecord();  
}
```

仔细看 Reducer 的处理逻辑是不是和 Combiner 的逻辑一样？注意最下面的逻辑判断部分，只是基于 Reducer 的输出结果，进行好友的逻辑判断：计数数值为 0 的才输出。

//重写 reduce()方法，逐条处理记录

```
public void reduce(Record key, Iterator<Record> values,  
    TaskContext context) throws IOException {  
    int count = 0;  
    while (values.hasNext()) {  
        Record val = values.next();  
        if( 0 == (Long)val.get(0) ) {  
            count = 0;  
            break;  
        }  
        count += (Long)val.get(0);  
    }  
    if(count > 0) {  
        sum.set(count);  
        String user=key.get(0).toString();  
        String[] users = user.split(" ");  
        user1.set(users[0]);  
        user2.set(users[1]);  
        result.set(0, user1);  
        result.set(1, user2);  
        result.set(2, sum);  
        context.write(result);  
    }  
}
```

(9) 完成 Driver 的处理（可参考 C:\ODPS_DEMO\resources\05-MR\FriendDriver.java）

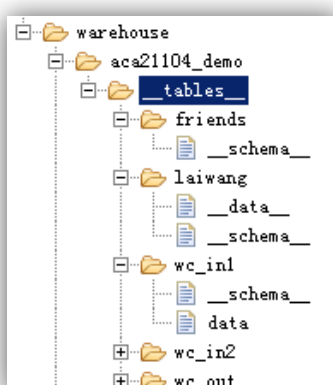
通过 Driver 完成 Job 的配置、注册和运行。

//通过 Driver 类进行 Job 的配置、管理和运行

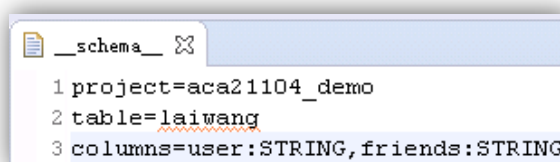
```
job.setMapOutputKeySchema(SchemaUtils.fromString("friends:string"));
job.setMapOutputValueSchema(SchemaUtils.fromString("cnt:bigint"));
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(),
    job);
outputUtils.addTable(TableInfo.builder().tableName(args[1]).build()
    ,job);
job.setMapperClass(FriendMapper.class);
job.setCombinerClass(FriendCombiner.class);
job.setReducerClass(FriendReducer.class);
RunningJob rj = JobClient.runJob(job);
rj.waitForCompletion();
```

(10) 本地模式测试该 MR:

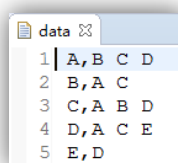
首先去增加两个本地测试用表，分别为 laiwang（输入表）、friends（输出表），两个本地表都位于 aca21104_demo 下的 __tables__ 下:



其中，laiwang 的表结构如下:

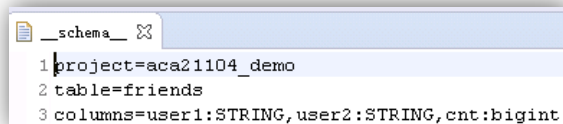


数据内容如下:



	A	B	C	D
1	A	B	C	D
2	B	A	C	
3	C	A	B	D
4	D	A	C	E
5	E	D		

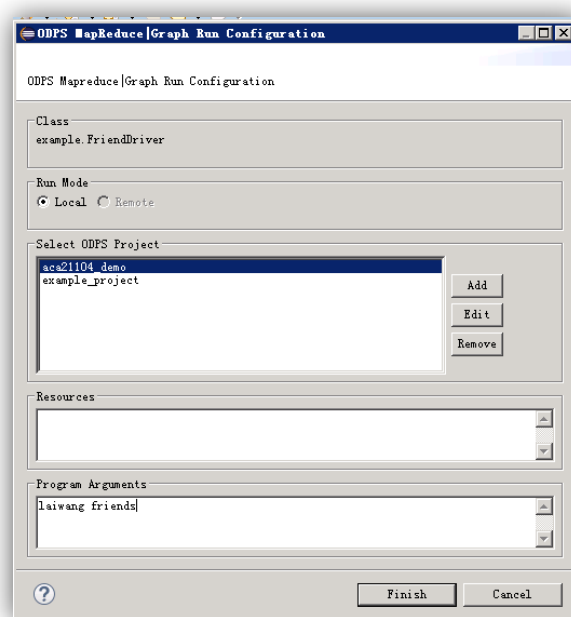
输出表 friends 没有数据，表结构如下：



```

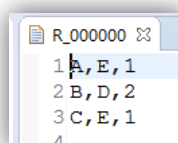
1 project=aca21104_demo
2 table=friends
3 columns=user1:STRING,user2:STRING,cnt:bigint
    
```

以 ODPS MapReduce 方式运行，在弹出窗口中选择 aca21104_demo，输入参数填入 laiwang friends:



操作不熟悉的同学，可以参考 2.13.2 中的本地模式测试的详细步骤。

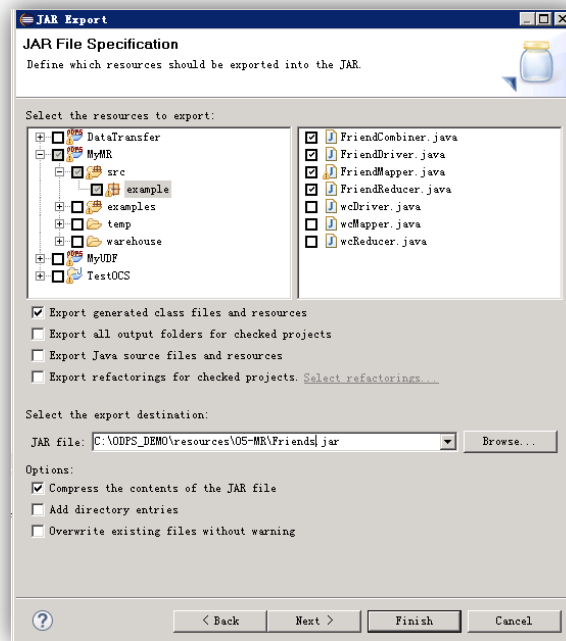
运行成功后，去检查一下输出表 friends 下的内容：



	A	B	C	D
1	A	E	1	
2	B	D	2	
3	C	E	1	
4				

输出结果符合预期，测试成功结束。

(11) 导出成 Jar 包，名字为：Friends.jar



(12) 将 jar 包作为资源上传到 odps:

切换至 jar 包所在目录，进入 odpscmd 交互模式：

```
cd C:\ODPS_DEMO\resources\05-MR\
odpscmd
```

上传资源：

```
add jar Friends.jar -f;
```

准备测试数据：

```
create table friends_list (userid string, friends string);
tunnel upload friends_list.csv friends_list;
create table friends_rec(userA string, userB string, cnt bigint);
```

执行 jar 包：

```
jar -resources Friends.jar -classpath Friends.jar example.FriendDriver
friends_list friends_rec;
```


2.13.4 实验：血缘分析

- (1) 继续使用 Java Project: myMR
- (2) 新增一个 ODPS Mapper 类，名字为 FamilyMapper
- (3) 新增一个 ODPS Reducer 类，名字为 FamilyReducer
- (4) 新增一个 ODPS Driver 类，名字为 FriendDriver
- (5) 完成 FamilyMapper 业务逻辑处理

可参考 C:\ODPS_DEMO\resources\05-MR\FamilyMapper.java

//重写 setup()方法，初始化

```
public void setup(TaskContext context) throws IOException {
    child = context.createMapOutputKeyRecord();
    parent = context.createMapOutputValueRecord();
}
```

//重写 map()方法，逐条处理记录

```
.....
String childName = record.get(0).toString();
String parentName = record.get(1).toString();
//left table: <key,value> = <child,parent>
child.set(new Object[]{childName});
parent.set(new Object[]{parentName + LEFT});
context.write(child, parent);
// right table: <key,value> = <parent, child>
child.set(new Object[]{childName + RIGHT});
parent.set(new Object[]{parentName});
context.write(parent, child);
.....
```

- (6) 完成 FamilyReducer 类的业务逻辑处理

可参考脚本 C:\ODPS_DEMO\resources\05-MR\FamilyReducer.java

//重写 setup()方法，初始化

```
public void setup(TaskContext context) throws IOException {
    result = context.createOutputRecord();
}
```

处理逻辑实现起来貌似麻烦，其实很简单，就是根据 Mapper 的结果，去继续寻找父节点的父节点或者去寻找子节点的子节点。简单来讲，把所有有血缘关系的人做成一条链路，这个链路的长度不限于 3 个节点，最后，遍历每一个链路，将同一链路上相连的任意三个节点输出即可。

//重写 map()方法，逐条处理记录

```
public void reduce(Record key, Iterator<Record> values, TaskContext context)
    throws IOException {
    childArr.clear();
    grandpaArr.clear();
    while (values.hasNext()) {
        Record val = values.next();
        String value = val.get(0).toString();
        if(value.endsWith(LEFT)) {
            grandpaArr.add(value.substring(0, value.length()-1));
        }
        else if(value.endsWith(RIGHT)){
            childArr.add(value.substring(0, value.length()-1));
        }
        else {
            throw new IOException("invalid value:" + value + ",
                key:" + key.toString());
        }
    }
    for(int i=0; i<childArr.size(); ++i) {
        for(int j=0; j<grandpaArr.size(); ++j) {
            result.set(0, childArr.get(i));
            result.set(1, key.get(0));
            result.set(2, grandpaArr.get(j));
            context.write(result);
        }
    }
}
```

(7) 在 FamilyDriver 中进行 job 的配置、管理和运行

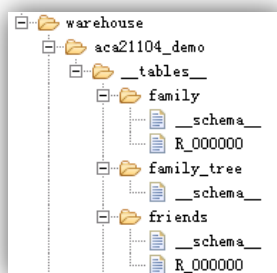
可参考 C:\ODPS_DEMO\resources\05-MR\FamilyDriver.java

//通过 Driver 类进行 Job 的配置、管理和运行

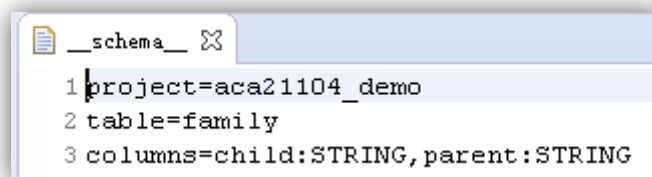
```
job.setMapOutputKeySchema(SchemaUtils.fromString("child:string"));
job.setMapOutputValueSchema(SchemaUtils.fromString("parent:string"));
InputUtils.addTable(TableInfo.builder().tableName(args[0]).build(),
    job);
OutputUtils.addTable(TableInfo.builder().tableName(args[1]).build(),
    job);
job.setMapperClass(FamilyMapper.class);
job.setReducerClass(FamilyReducer.class);
RunningJob rj = JobClient.runJob(job);
rj.waitForCompletion();
```

(8) 本地模式测试该 MR:

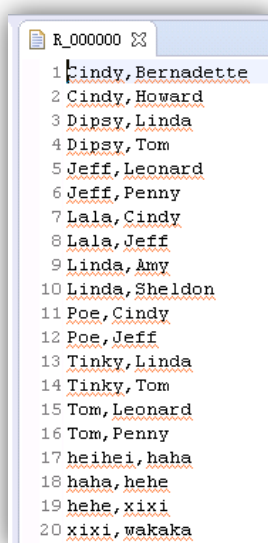
首先去增加两个本地测试用表，分别为 family（输入表）、family_tree（输出表），两个本地表都位于 aca21104_demo 下的 __tables__ 下：



其中，family 的表结构如下：

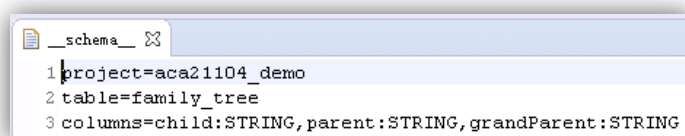


family 中的测试数据如下：



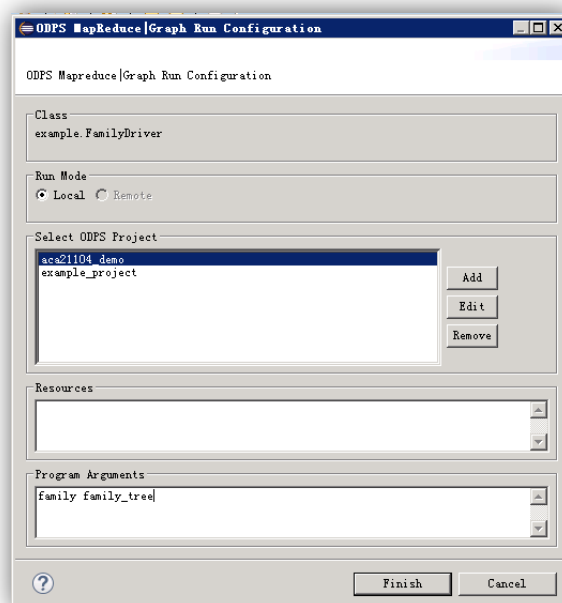
```
R_000000
1 Cindy, Bernadette
2 Cindy, Howard
3 Dipsy, Linda
4 Dipsy, Tom
5 Jeff, Leonard
6 Jeff, Penny
7 Lala, Cindy
8 Lala, Jeff
9 Linda, Amy
10 Linda, Sheldon
11 Poe, Cindy
12 Poe, Jeff
13 Tinky, Linda
14 Tinky, Tom
15 Tom, Leonard
16 Tom, Penny
17 heihei, haha
18 haha, hehe
19 hehe, xixi
20 xixi, wakaka
```

family_tree 的表结构如下:



```
_schema_
1 project=aca21104_demo
2 table=family_tree
3 columns=child:STRING,parent:STRING,grandParent:STRING
```

测试数据准备好后, 以 ODPS MapReduce 的模式运行 FamilyDriver.java, 在弹出的配置窗口中选择 ODPS project 为 aca21104_demo, 输入参数为 family family_tree:

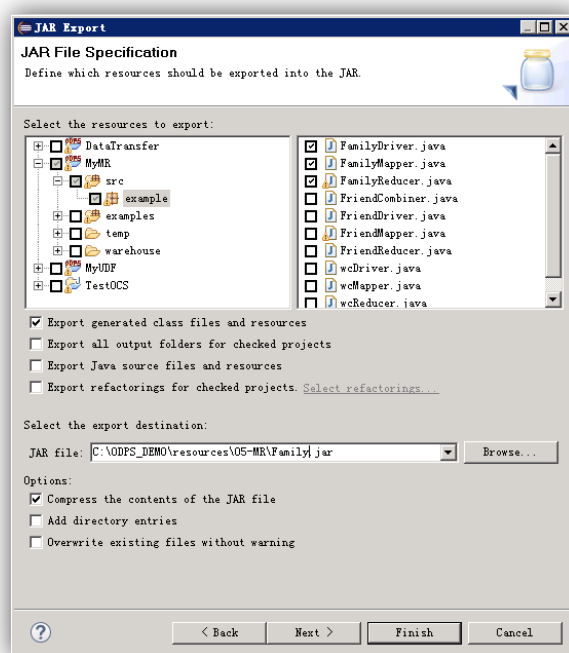


运行结束后, 检查结果:

```
R_000000
1 Poe, Cindy, Bernadette
2 Poe, Cindy, Howard
3 Lala, Cindy, Bernadette
4 Lala, Cindy, Howard
5 Lala, Jeff, Leonard
6 Lala, Jeff, Penny
7 Poe, Jeff, Leonard
8 Poe, Jeff, Penny
9 Dipsy, Linda, Amy
10 Dipsy, Linda, Sheldon
11 Tinky, Linda, Amy
12 Tinky, Linda, Sheldon
13 Dipsy, Tom, Leonard
14 Dipsy, Tom, Penny
15 Tinky, Tom, Leonard
16 Tinky, Tom, Penny
17 hehe1, haha, hehe
18 haha, hehe, xixi
19 hehe, xixi, wakaka
```

结果符合预期，测试成功结束。

(9) 导出成 Jar 包，名字为: Friends.jar



(10) 将 jar 包作为资源上传到 odps:

切换至 jar 包所在目录，进入 odpscmd 交互模式:

```
cd C:\ODPS_DEMO\resources\05-MR\
odpscmd
```

上传资源:

```
add jar Family.jar -f;
```

准备测试数据:

```
create table family (child string, parent string);  
tunnel upload family.csv aca21104_demo.family;  
create table family_tree(child string, parent string, grandparent string);
```

执行 jar 包:

```
jar -resources Family.jar -classpath Family.jar example.FamilyDriver  
family family_tree;
```

2.14 安全与权限

2.14.1 实验: 准备测试环境

该实验需要和其他同学配合完成, 请找到一个可以和你互相配合的同学, 下文中会提到 A 用户、B 用户, 其中 A 用户即为你本人使用的阿里云的账号, B 用户为和你配合的同学的账号。个别实验可能还需要第三个同学的配合, 当我们提到 C 用户的时候, 即需要使用第三个同学的阿里云账号。

2.14.2 实验: A 用户授权给 B 用户

查看当前项目中用户和权限情况:

```
show grants;
```

将用户 B 加入到当前项目空间中来:

```
add user ALIYUN$trainingdemo_try@aliyun-test.com;
```

将表 dual 的访问权限赋予用户 B:

```
grant select, describe on table dual to user ALIYUN$trainingdemo_try@aliyun-test.com;
```

查看 B 用户拥有的权限:

```
show grants for ALIYUN$trainingdemo_try@aliyun-test.com;
```

使用用户 B 去访问表 aca21104_demo.dual:

```
select * from aca21104_demo.dual;
```

用户 A 将用户 B 从项目中移除:

```
remove user ALIYUN$trainingdemo_try@aliyun-test.com;
```

使用用户 B 去访问表 aca21104_demo.dual, 此时访问报错:

```
select * from aca21104_demo.dual;
```

查看 B 用户拥有的权限:

```
show grants for ALIYUN$trainingdemo_try@aliyun-test.com;
```

用户 A 将用户 B 加入到当前项目空间中来:

```
add user ALIYUN$trainingdemo_try@aliyun-test.com;
```

使用用户 B 去访问表 aca21104_demo.dual, 成功, 原有权限自动生效:

```
select * from aca21104_demo.dual;
```

用户 A 收回 B 用户对 dual 表的读的权限:

```
revoke select,describe on table dual from user ALIYUN$trainingdemo_try@aliyun-test.com;
```

2.14.3 实验: 角色管理与授权

用户 A 新建一个角色 reader:

```
create role reader;
```

用户 A 赋予角色 reader 几张表的读权限:

```
grant describe, select on table dual to role reader;  
grant describe, select on table t_test to role reader;
```

用户 A 查看角色的权限:

```
desc role reader;
```

用户 A 查看 B 用户拥有的权限:

```
show grants for ALIYUN$trainingdemo_try@aliyun-test.com;
```

用户 A 将角色 reader 赋予用户 B:

```
grant reader to ALIYUN$trainingdemo_try@aliyun-test.com;
```

用户 A 查看 B 用户拥有的权限:

```
show grants for ALIYUN$trainingdemo_try@aliyun-test.com;
```

用户 A 赋予角色 reader 另外几张表的读权限:

```
grant describe, select on table t_tunnel to role reader;  
grant describe, select on table t_tunnel_p to role reader;
```

用户 A 查看角色的权限和使用情况:

```
desc role reader;
```

用户 A 查看 B 用户拥有的权限:

```
show grants for ALIYUN$trainingdemo_try@aliyun-test.com;
```

用户 A 删除角色 reader (会报错, 删除失败):

```
drop role reader;
```

用户 A 移除用户 B (会报错, 移除失败):

```
remove user ALIYUN$trainingdemo_try@aliyun-test.com;
```


用户 A 查看角色的权限和使用情况：

```
desc role reader;
```

用户 A 从用户 B 收回角色：

```
revoke reader from ALIYUN$trainingdemo_try@aliyun-test.com;
```

用户 A 删除角色 reader：

```
drop role reader;
```

用户 A 移除用户 B：

```
remove user ALIYUN$trainingdemo_try@aliyun-test.com;
```

2. 14. 4 实验：鉴权模型查看与管理

用户 A 查看当前项目的鉴权模型：

```
show SecurityConfiguration;
```

用户 A 赋予用户 B 建表的权限：

```
add user ALIYUN$trainingdemo_try@aliyun-test.com;  
grant createtable on project aca21104_demo to user ALIYUN$trainingdemo_try  
@aliyun-test.com;
```

用户 B 建表：

```
create table aca21104_demo.t_test_sg (id int);  
desc aca21104_demo.t_test_sg;  
select count(*) from aca21104_demo.t_test_sg;
```

用户 A 修改鉴权模型，将 ObjectCreatorHasAccessPermission 改为 false：

```
set ObjectCreatorHasAccessPermission=false;  
show SecurityConfiguration;
```

用户 B 访问 aca21104_demo.t_test_sg(报错，权限不足)：

```
desc aca21104_demo.t_test_sg;  
select count(*) from aca21104_demo.t_test_sg;
```

用户 B 删除 aca21104_demo.t_test_sg(报错, 权限不足):

```
drop table aca21104_demo.t_test_sg;
```

用户 A 修改鉴权模型, 将 ObjectCreatorHasAccessPermission 改为 true:

```
set ObjectCreatorHasAccessPermission=true;  
show SecurityConfiguration;
```

用户 B 访问 aca21104_demo.t_test_sg:

```
desc aca21104_demo.t_test_sg;  
select count(*) from aca21104_demo.t_test_sg;
```

用户 B 删除 aca21104_demo.t_test_sg:

```
drop table aca21104_demo.t_test_sg;
```

用户 A 收回用户 B 建表的权限:

```
revoke createtable on project aca21104_demo from user ALIYUN$trainingdemo_  
try@aliyun-test.com;  
remove user ALIYUN$trainingdemo_try@aliyun-test.com;
```

2.14.5 实验: 基于标签的安全控制

用户 A 查看当前项目的鉴权模型:

```
show SecurityConfiguration;
```

如果 LabelSecurity 的值为 false, 则需要将它设置成 true:

```
set LabelSecurity=true;
```

增加用户 B 到当前项目 aca21104_demo, 并设置安全许可标签:

默认安全标签为 0, 我们将用户 B(ALIYUN\$trainingdemo_try@aliyun-test.com) 安全许可标签设置为 3

```
add user ALIYUN$trainingdemo_try@aliyun-test.com;  
set label 3 to user ALIYUN$trainingdemo_try@aliyun-test.com;  
grant select,describe on table t_tunnel to user ALIYUN$trainingdemo_try@aliyun-test.com;
```

用户 B 此时可以访问 aca21104_demo.t_tunnel:

```
select * from aca21104_demo.t_tunnel;
```

用户 A 将 t_tunnel 的敏感等级提高成 4:

```
set label 4 to table t_tunnel;
```

用户 B 此时因安全等级低, 无法访问敏感等级高的数据 aca21104_demo.t_tunnel:

```
select * from aca21104_demo.t_tunnel;
```

数据敏感级别可以到列: 用户 A 将 t_tunnel 中的 id 字段的敏感度调整到 3:

```
set label 3 to table t_tunnel(id);
```

用户 B 此时可以访问 aca21104_demo.t_tunnel 中敏感级别不大于 3 的列 id:

```
select id from aca21104_demo.t_tunnel;
```

用户 A 可以设置对低权限用户 B 进行临时授权可以访问高敏感级别的表 t_tunnel_p:

```
set label 3 to user ALIYUN$trainingdemo_try@aliyun-test.com;  
grant select,describe on table t_tunnel_p to user ALIYUN$trainingdemo_try@aliyun-test.com;  
set label 5 to table t_tunnel_p(name);  
set label 4 to table t_tunnel_p(id);  
grant label 4 on table t_tunnel_p to user ALIYUN$trainingdemo_try@aliyun-test.com with exp 1;
```

用户 B 此时可以访问 aca21104_demo.t_tunnel 中敏感级别不大于 4 的所有列:

```
select id from aca21104_demo.t_tunnel_p;  
select name from aca21104_demo.t_tunnel_p;
```

用户 A 查看当前有权限访问 t_tunnel_p 的表的表的用户列表:

```
show label grants on table aca21104_demo.t_tunnel_p;
```

用户 A 查看用户 B 有权限访问的所有的表的列表:

```
show label grants on table aca21104_demo.t_tunnel_p;
```

用户 A 收回用户 B 权限, 并从项目中移除用户 B, 并复原项目的鉴权模型:

```
revoke describe,select on table t_tunnel from user ALIYUN$trainingdemo_try
@aliyun-test.com;
revoke describe,select on table t_tunnel_p from user ALIYUN$trainingdemo_t
ry@aliyun-test.com;
remove user ALIYUN$trainingdemo_try@aliyun-test.com;
set LabelSecurity=false;
show SecurityConfiguration;
```

2. 14. 6 实验: 跨项目空间的资源分享

用户 A 创建 package, 并将表 t_tunnel 以及表 t_tunnel_p 的访问权限添加到该 package:

```
create package pk_tunnel_read;
add table t_tunnel to package pk_tunnel_read with privileges select;
add table t_tunnel_p to package pk_tunnel_read with privileges describe;
```

用户 A 将 pk_tunnel_read 赋给用户 B 所在的项目 secret_garden:

```
allow project secret_garden to install package pk_tunnel_read;
```

用户 B 查看所在项目空间可用的 package:

```
show packages;
```

用户 B 安装 pk_tunnel_read:

```
install package aca21104_demo.pk_tunnel_read;
show packages;
desc package aca21104_demo.pk_tunnel_read;
```

用户 B 访问 package 中包含的资源:

```
select * from aca21104_demo.t_tunnel;  
desc aca21104_demo.t_tunnel_p;
```

用户 B 可以把 package 赋给当前项目中的其他用户。

用户 B 卸载 package:

```
uninstall package aca21104_demo.pk_tunnel_read;
```

用户 A 删除 package:

```
drop package pk_tunnel_read;
```

2. 14. 7 实验：项目空间保护

用户 A 查看当前项目的鉴权模型，确认目前的 ProjectProtection 处于 false 状态:

```
show SecurityConfiguration;
```

增加用户 B 到当前项目 aca21104_demo，并赋予表 t_tunnel 的读写权限:

```
add user ALIYUN$trainingdemo_try@aliyun-test.com;  
grant all on table t_tunnel to user ALIYUN$trainingdemo_try@aliyun-test.co  
m;  
grant select on table dual to user ALIYUN$trainingdemo_try@aliyun-test.co  
m;
```

用户 B 操作表 t_tunnel: 可以读取记录，或者插入数据

```
select * from aca21104_demo.t_tunnel;  
insert into table aca21104_demo.t_tunnel select -1, 'from_sg' from aca2110  
4_demo.dual;
```

用户 B 甚至可以把表中的数据搬到本地的项目中来:

```
create table iris as select * from aca21104_demo.t_tunnel;
```

用户 A 为了防止数据流出，打开项目保护选项: ProjectProtection:

```
set ProjectProtection=true;
```

用户 B 对 t_tunnel 的操作都被禁止，需要联系 aca21104_demo 的 owner:

```
select * from aca21104_demo.t_tunnel;
insert into table aca21104_demo.t_tunnel select -1, 'from_sg' from aca21104_demo.dual;
create table iris_again as select * from aca21104_demo.t_tunnel;
```

用户 A 为用户 B 设置例外:

```
set ProjectProtection=true with exception c:\pf_try;
```

其中, pf_try 为一个授权策略文件, 内容如下:

```
{
  "version": "1",
  "statement": [
    {
      "effect": "Allow",
      "principal": "ALIYUN$trainingdemo_try@aliyun-test.com",
      "action": ["odps:select"],
      "resource": "acs:odps:*:projects/aca21104_demo/tables/t_tunnel",
      "condition": {
        "stringEquals": {
          "odps:TaskType": ["DT", "SQL"]
        }
      }
    }
  ]
}
```

用户 B 可以对表 aca21104_demo.t_tunnel 进行正常操作:

```
select * from aca21104_demo.t_tunnel;
create table iris_again as select * from aca21104_demo.t_tunnel;
```

也可以通过另一种方式实现。

用户 A 设置无例外项目保护:

```
set ProjectProtection=true;
```

2.14.8 实验: 访问策略语言

用户 A 授权用户 B 只能在” 2016-11-11T23:59:59Z” 这个时间点之前, 只能从某些指定的 IP 或者 IP 段来提交请求, 只允许该用户在项目空间 aca21104_demo 中执行 CreateInstance,

CreateTable 和 List 操作，禁止该用户删除 aca21104_demo 下的任何 table。则策略语言对应的文件内容如下：

```
{
  "Version": "1",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "ALIYUN$trainingdemo_try@aliyun-test.com",
      "Action": ["odps:CreateTable", "odps:CreateInstance", "odps:List"],
      "Resource": "acs:odps:*:projects/aca21104_demo",
      "Condition": {
        "DateLessThan": {
          "acs:CurrentTime": "2016-11-11T23:59:59Z"
        },
        "IpAddress": {
          "acs:SourceIp": "10.32.180.0/23"
        }
      }
    },
    {
      "Effect": "Deny",
      "Principal": "ALIYUN$trainingdemo_try@aliyun-test.com",
      "Action": "odps:Drop",
      "Resource": "acs:odps:*:projects/aca21104_demo/tables/*"
    }
  ]
}
```

用户 A 使用如下命令，使该策略生效：

```
put policy <pf.json>
```

用户 B 进行一些操作来验证该策略是否真正生效。

三、实验总结

无