**上海联通大数据平台**

**Hive开发规范**

(v1.0.4)

**上海联通信息化事业部**

目录

[1 更改记录 3](#_Toc420428408)

[2 文档概述 4](#_Toc420428409)

[3 Hive开发基础原则 4](#_Toc420428410)

[3.1 表分区键命名原则 4](#_Toc420428411)

[3.2 Hive表文件类型选择原则 4](#_Toc420428412)

[3.3 Hive表压缩使用原则 4](#_Toc420428413)

[3.4 Hive字段分隔符选取原则 5](#_Toc420428414)

[4 建表规范 5](#_Toc420428415)

[4.1 接口表 5](#_Toc420428416)

[4.2 内部数据加工表 6](#_Toc420428417)

[4.2.1 月分区表 6](#_Toc420428418)

[4.2.2 月、日分区表 7](#_Toc420428419)

[4.2.3 日期分区表 8](#_Toc420428420)

[4.3 Hive表数据在HDFS存放位置 9](#_Toc420428421)

[5 Hive脚本开发规范 9](#_Toc420428422)

[5.1 目录规划 9](#_Toc420428423)

[5.2 脚本模板 10](#_Toc420428424)

[5.2.1 配置文件 10](#_Toc420428425)

[5.2.2 业务逻辑脚本 11](#_Toc420428426)

[6 Hive UDF自定义函数开发规范 14](#_Toc420428427)

[6.1 开发步骤 14](#_Toc420428428)

[6.2 开发工具 14](#_Toc420428429)

[6.3 UDF函数示例 15](#_Toc420428430)

[6.4 Hive UDF函数(逐步补充) 17](#_Toc420428431)

[6.4.1 UDF函数列表 17](#_Toc420428432)

[6.4.2 UDF函数说明 18](#_Toc420428433)

[7 附录一：Hive基本语法 19](#_Toc420428434)

[7.1 参考文档1 19](#_Toc420428435)

[7.1.1 Create Table 19](#_Toc420428436)

[7.1.2 Drop Table 20](#_Toc420428437)

[7.1.3 Alter Table 20](#_Toc420428438)

[7.1.4 Loading files into table 22](#_Toc420428439)

[7.1.5 SELECT 23](#_Toc420428440)

[7.1.6 Join 25](#_Toc420428441)

[7.2 参考文档2 29](#_Toc420428442)

[7.2.1 DDL 操作 29](#_Toc420428443)

[7.2.2 DML 操作:元数据存储 35](#_Toc420428444)

[7.2.3 DQL 操作:数据查询SQL 37](#_Toc420428445)

[7.2.4 从SQL到HiveQL应转变的习惯 42](#_Toc420428446)

[7.2.5 实际示例 44](#_Toc420428447)

[8 附录二：Hive函数 47](#_Toc420428448)

[8.1 关系运算 47](#_Toc420428449)

[8.1.1 等值比较: = 47](#_Toc420428450)

[8.1.2 不等值比较: <> 47](#_Toc420428451)

[8.1.3 小于比较: < 47](#_Toc420428452)

[8.1.4 小于等于比较: <= 47](#_Toc420428453)

[8.1.5 大于比较: > 48](#_Toc420428454)

[8.1.6 大于等于比较: >= 48](#_Toc420428455)

[8.1.7 空值判断: IS NULL 48](#_Toc420428456)

[8.1.8 非空判断: IS NOTNULL 49](#_Toc420428457)

[8.1.9 LIKE比较: LIKE 49](#_Toc420428458)

[8.1.10 JAVA的LIKE操作: RLIKE 49](#_Toc420428459)

[8.1.11 REGEXP操作: REGEXP 50](#_Toc420428460)

[8.2 数学运算 50](#_Toc420428461)

[8.2.1 加法操作: + 50](#_Toc420428462)

[8.2.2 减法操作: - 50](#_Toc420428463)

[8.2.3 乘法操作: \* 51](#_Toc420428464)

[8.2.4 除法操作: / 51](#_Toc420428465)

[8.2.5 取余操作: % 51](#_Toc420428466)

[8.2.6 位与操作: & 52](#_Toc420428467)

[8.2.7 位或操作: | 52](#_Toc420428468)

[8.2.8 位异或操作: ^ 52](#_Toc420428469)

[8.2.9 位取反操作: ~ 52](#_Toc420428470)

[8.3 逻辑运算 53](#_Toc420428471)

[8.3.1 逻辑与操作: AND 53](#_Toc420428472)

[8.3.2 逻辑或操作: OR 53](#_Toc420428473)

[8.3.3 逻辑非操作: NOT 53](#_Toc420428474)

[8.4 数值计算 54](#_Toc420428475)

[8.4.1 取整函数: round 54](#_Toc420428476)

[8.4.2 指定精度取整函数: round 54](#_Toc420428477)

[8.4.3 向下取整函数: floor 54](#_Toc420428478)

[8.4.4 向上取整函数: ceil 54](#_Toc420428479)

[8.4.5 向上取整函数: ceiling 55](#_Toc420428480)

[8.4.6 取随机数函数: rand 55](#_Toc420428481)

[8.4.7 自然指数函数: exp 55](#_Toc420428482)

[8.4.8 以10为底对数函数: log10 56](#_Toc420428483)

[8.4.9 以2为底对数函数: log2 56](#_Toc420428484)

[8.4.10 对数函数: log 56](#_Toc420428485)

[8.4.11 幂运算函数: pow 56](#_Toc420428486)

[8.4.12 幂运算函数: power 57](#_Toc420428487)

[8.4.13 开平方函数: sqrt 57](#_Toc420428488)

[8.4.14 二进制函数: bin 57](#_Toc420428489)

[8.4.15 十六进制函数: hex 57](#_Toc420428490)

[8.4.16 反转十六进制函数: unhex 58](#_Toc420428491)

[8.4.17 进制转换函数: conv 58](#_Toc420428492)

[8.4.18 绝对值函数: abs 58](#_Toc420428493)

[8.4.19 正取余函数: pmod 58](#_Toc420428494)

[8.4.20 正弦函数: sin 59](#_Toc420428495)

[8.4.21 反正弦函数: asin 59](#_Toc420428496)

[8.4.22 余弦函数: cos 59](#_Toc420428497)

[8.4.23 反余弦函数: acos 59](#_Toc420428498)

[8.4.24 positive函数: positive 60](#_Toc420428499)

[8.4.25 negative函数: negative 60](#_Toc420428500)

[8.5 日期函数 60](#_Toc420428501)

[8.5.1 UNIX时间戳转日期函数:from\_unixtime 60](#_Toc420428502)

[8.5.2 获取当前UNIX时间戳函数:unix\_timestamp 60](#_Toc420428503)

[8.5.3 日期转UNIX时间戳函数:unix\_timestamp 61](#_Toc420428504)

[8.5.4 指定格式日期转UNIX时间戳函数:unix\_timestamp 61](#_Toc420428505)

[8.5.5 日期时间转日期函数:to\_date 61](#_Toc420428506)

[8.5.6 日期转年函数: year 61](#_Toc420428507)

[8.5.7 日期转月函数: month 62](#_Toc420428508)

[8.5.8 日期转天函数: day 62](#_Toc420428509)

[8.5.9 日期转小时函数: hour 62](#_Toc420428510)

[8.5.10 日期转分钟函数: minute 62](#_Toc420428511)

[8.5.11 日期转秒函数: second 63](#_Toc420428512)

[8.5.12 日期转周函数:weekofyear 63](#_Toc420428513)

[8.5.13 日期比较函数: datediff 63](#_Toc420428514)

[8.5.14 日期增加函数: date\_add 63](#_Toc420428515)

[8.5.15 日期减少函数: date\_sub 63](#_Toc420428516)

[8.6 条件函数 64](#_Toc420428517)

[8.6.1 If函数: if 64](#_Toc420428518)

[8.6.2 非空查找函数: COALESCE 64](#_Toc420428519)

[8.6.3 条件判断函数：CASE 64](#_Toc420428520)

[8.6.4 条件判断函数：CASE 64](#_Toc420428521)

[8.7 字符串函数 65](#_Toc420428522)

[8.7.1 字符串长度函数：length 65](#_Toc420428523)

[8.7.2 字符串反转函数：reverse 65](#_Toc420428524)

[8.7.3 字符串连接函数：concat 65](#_Toc420428525)

[8.7.4 带分隔符字符串连接函数：concat\_ws 65](#_Toc420428526)

[8.7.5 字符串截取函数：substr,substring 66](#_Toc420428527)

[8.7.6 字符串截取函数：substr,substring 66](#_Toc420428528)

[8.7.7 字符串转大写函数：upper,ucase 66](#_Toc420428529)

[8.7.8 字符串转小写函数：lower,lcase 67](#_Toc420428530)

[8.7.9 去空格函数：trim 67](#_Toc420428531)

[8.7.10 左边去空格函数：ltrim 67](#_Toc420428532)

[8.7.11 右边去空格函数：rtrim 67](#_Toc420428533)

[8.7.12 正则表达式替换函数：regexp\_replace 67](#_Toc420428534)

[8.7.13 正则表达式解析函数：regexp\_extract 68](#_Toc420428535)

[8.7.14 URL解析函数：parse\_url 68](#_Toc420428536)

[8.7.15 json解析函数：get\_json\_object 69](#_Toc420428537)

[8.7.16 空格字符串函数：space 69](#_Toc420428538)

[8.7.17 重复字符串函数：repeat 69](#_Toc420428539)

[8.7.18 首字符ascii函数：ascii 69](#_Toc420428540)

[8.7.19 左补足函数：lpad 70](#_Toc420428541)

[8.7.20 右补足函数：rpad 70](#_Toc420428542)

[8.7.21 分割字符串函数: split 70](#_Toc420428543)

[8.7.22 集合查找函数:find\_in\_set 70](#_Toc420428544)

[8.8 集合统计函数 71](#_Toc420428545)

[8.8.1 个数统计函数: count 71](#_Toc420428546)

[8.8.2 总和统计函数: sum 71](#_Toc420428547)

[8.8.3 平均值统计函数: avg 71](#_Toc420428548)

[8.8.4 最小值统计函数: min 72](#_Toc420428549)

[8.8.5 最大值统计函数: max 72](#_Toc420428550)

[8.8.6 非空集合总体变量函数:var\_pop 72](#_Toc420428551)

[8.8.7 非空集合样本变量函数:var\_samp 72](#_Toc420428552)

[8.8.8 总体标准偏离函数:stddev\_pop 72](#_Toc420428553)

[8.8.9 样本标准偏离函数:stddev\_samp 73](#_Toc420428554)

[8.8.10 中位数函数:percentile 73](#_Toc420428555)

[8.8.11 中位数函数:percentile 73](#_Toc420428556)

[8.8.12 近似中位数函数:percentile\_approx 73](#_Toc420428557)

[8.8.13 近似中位数函数:percentile\_approx 73](#_Toc420428558)

[8.8.14 直方图:histogram\_numeric 74](#_Toc420428559)

[8.9 复合类型构建操作 74](#_Toc420428560)

[8.9.1 Map类型构建: map 74](#_Toc420428561)

[8.9.2 Struct类型构建: struct 74](#_Toc420428562)

[8.9.3 array类型构建: array 74](#_Toc420428563)

[8.10 复杂类型访问操作 75](#_Toc420428564)

[8.10.1 array类型访问: A[n] 75](#_Toc420428565)

[8.10.2 map类型访问: M[key] 75](#_Toc420428566)

[8.10.3 struct类型访问: S.x 75](#_Toc420428567)

[8.11 复杂类型长度统计函数 76](#_Toc420428568)

[8.11.1 Map类型长度函数: size(Map<K.V>) 76](#_Toc420428569)

[8.11.2 array类型长度函数: size(Array<T>) 76](#_Toc420428570)

[8.11.3 类型转换函数 76](#_Toc420428571)

[9 附录三：常见问题解决方法 76](#_Toc420428572)

[9.1 问题1 76](#_Toc420428573)

[9.1.1 错误信息 76](#_Toc420428574)

[9.1.2 解决办法： 77](#_Toc420428575)

[9.2 问题2 77](#_Toc420428576)

[9.2.1 错误信息 77](#_Toc420428577)

[9.2.2 解决办法 77](#_Toc420428578)

[9.3 问题3 77](#_Toc420428579)

[9.3.1 错误信息 77](#_Toc420428580)

[9.3.2 解决办法 77](#_Toc420428581)

[9.4 问题4 78](#_Toc420428582)

[9.4.1 错误信息 78](#_Toc420428583)

[9.4.2 解决办法 78](#_Toc420428584)

[9.5 问题5 78](#_Toc420428585)

[9.5.1 错误信息 78](#_Toc420428586)

[9.5.2 解决办法 78](#_Toc420428587)

# 更改记录

|  |  |  |
| --- | --- | --- |
| **变更内容** | **变更人** | **变更时间** |
| Hive开发规范1.0.0成稿 | 平晓刚 | 20150529 |
| Hive开发规范1.0.1，增加5.2数据加载模板 | 平晓刚 | 20150706 |
| Hive开发规范1.0.2，修改5.4.2.1 基础逻辑模板，修改4.1 接口表建表模板 | 平晓刚 | 20150707 |
| 增加4.1标注释规范 | 平晓刚 | 20150708 |
| 修改3.1表分区键命名原则，增加余数分区命名规范 | 平晓刚 | 20150723 |

# 文档概述

本文定义了在大数据平台项目实施中，HIVE相关的开发规范，其中包括：基本的开发原则、建表规范、HIVE脚本开发规范、Hive UDF自定义函数开发规范等。

大数据平台HIVE数据加工脚本开发开发过程中，要求严格遵守本规范要求，完成脚本开发。

# Hive开发基础原则

## 表分区键命名原则

Hive开发过程中，数据表需要设置为分区表，分区键统一使用如下命名规则：

|  |  |  |
| --- | --- | --- |
| **分区键含义** | **分区键命名** | **备注说明** |
| 月分区 | month\_part | 六位年月：yyyymm |
| 日分区 | day\_part | 两位日期：dd |
| 日期分区 | date\_part | 八位日期：yyyymmdd |
| 地域分区 | area\_part | 区县编码 |
| 余数分区 | ID\_PART | 常用为日期的除以保存周期的余数 |

【**注意**】：**创建分区键的时候分区键不能和表字段名相同。**

## Hive表文件类型选择原则

接口层数据表和导出数据表，采用**TEXTFILE**文件类型；

数据加工过程中的数据表，采用**RCFILE**文件类型。

## Hive表压缩使用原则

Hive数据仓库在配置文件中，执行MapReduce作业生成的数据结果，会按照配置文件中的文件压缩方式进行压缩，此处我们使用snappy，配合表文件类型使用RCFILE。

**值得注意的是：**

1. 通过LOAD命令、hdfs dfs –put命令加载到HIVE数据库中的文件并不压缩，所以需要使用TEXTFILE。即使接口文件本身为压缩文件，HIVE也可正常使用数据；
2. 用于导出数据进行数据服务的表，本身数据是压缩类型，导出后需要解压缩，为了方便操作，用于导出的数据表进行数据加工前，可通过指定参数：

set hive.exec.compress.output=false，指定参数后，数据表中加工出来的数据不进行压缩，可以直接使用LOAD命令、hdfs dfs –get命令进行数据文件导出。

## Hive字段分隔符选取原则

在Hive建表过程中，需要指定字段分隔符采用什么分隔符，本次大数据平台建设，主要有如下两种情况：

1. 数据接口表：字段分割符采用接口规范中指定的源文件中的字段分隔符
2. 内部数据加工结果表：表的字段分割符默认使用'\001'

# 建表规范

## 表注释

 表的注释都正确填写，如：

移动业务订购实例资料table;[MASTER\_FIELD:B域,LEVEL:DWD层,DATA\_FIELD:产品域,DATASUB\_FIELD:订购实例]

其中，域命名（DATA\_FIELD）填写规则如下：

说明：代码为原ORACLE数据库中表名称第二段关键字内容，如：ODS\_**CDR**\_OCS\_RECHARGE\_D为详单域

|  |  |  |
| --- | --- | --- |
| 域 | 代码 | 说明 |
| 客户 | CUS | CUSTOMER,包含个人、组织客户信息；VIP、俱乐部等信息 |
| 用户 | USR | USER，包含订购实例细节，用户各种属性 |
| 产品 | PRD | PRODUCT，产品体系，包含产品编码，产品所包含的包、子产品、服务等各类属性 |
| 渠道 | CHL | CHANNEL，渠道，代理商，佣金等信息 |
| 营业 | BUS | BUSINESS，工单记录，营业记录 |
| 详单(使用) | CDR | Call Detail Records，各类业务的通话、使用记录 |
| 账务 | BIL | Billing,付费关系，出账、销账、欠费等账务相关 |
| 合作伙伴 | PRT | Partner，SP管理及其提供的增值业务等信息，以及SP结算相关数据 |
| 资源 | RES | Resource, 卡类、终端、基站等资源信息 |
| 市场营销 | MRT | Marketing, 市场营销，维系挽留等数据 |
| 公共域 | PUB | Public，地域管理，员工、部门信息以及一些其他信息 |

## 接口表

根据第2章 Hive开发基础原则中描述，接口表**文件类型定义为TEXTFILE**，**不建立分区**，数据分隔符**以接口文件中实际分隔符为准**。

**示例：**

CREATE TABLE src.gn\_table(

a1 string COMMENT 'a1',

a2 string COMMENT 'a2',

a3 string COMMENT 'a3',

a4 string COMMENT 'a4',

a5 string COMMENT 'a5',

a6 string COMMENT 'a6',

a7 string COMMENT 'a7'

)

COMMENT '表描述'

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001'

STORED AS TEXTFILE;

## 内部数据加工表

内部数据加工表，需要**建立分区**，文件类型**采用RCFILE**，**分隔符使用Hive默认的’\001’**。

### 月分区表

**示例：**

CREATE TABLE ODS.ODS\_GN\_TABLE(

a1 string COMMENT 'a1列注释',

a2 string COMMENT 'a2列注释',

a3 string COMMENT 'a3列注释',

a4 string COMMENT 'a4列注释',

a5 string COMMENT 'a5列注释',

a6 string COMMENT 'a6列注释',

a7 string COMMENT 'a7列注释',

a8 string COMMENT 'a8列注释',

a9 string COMMENT 'a9列注释',

a10 string COMMENT 'a10列注释',

a11 string COMMENT 'a11列注释',

a12 string COMMENT 'a12列注释',

a13 string COMMENT 'a13列注释',

a14 string COMMENT 'a14列注释',

a15 string COMMENT 'a15列注释',

a16 string COMMENT 'a16列注释',

a17 string COMMENT 'a17列注释',

a18 string COMMENT 'a18列注释',

a19 string COMMENT 'a19列注释',

a20 string COMMENT 'a20列注释',

a21 string COMMENT 'a21列注释',

a22 string COMMENT 'a22列注释',

a23 string COMMENT 'a23列注释',

a24 string COMMENT 'a24列注释',

a25 string COMMENT 'a25列注释',

a26 string COMMENT 'a26列注释',

a27 string COMMENT 'a27列注释')

COMMENT 'GN数据测试表'

PARTITIONED BY (MONTH\_PART STRING comment ‘月份分区’)

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001'

STORED AS RCFILE;

### 月、日分区表

**示例：**

CREATE TABLE ODS.ODS\_GN\_TABLE(

a1 string COMMENT 'a1列注释',

a2 string COMMENT 'a2列注释',

a3 string COMMENT 'a3列注释',

a4 string COMMENT 'a4列注释',

a5 string COMMENT 'a5列注释',

a6 string COMMENT 'a6列注释',

a7 string COMMENT 'a7列注释',

a8 string COMMENT 'a8列注释',

a9 string COMMENT 'a9列注释',

a10 string COMMENT 'a10列注释',

a11 string COMMENT 'a11列注释',

a12 string COMMENT 'a12列注释',

a13 string COMMENT 'a13列注释',

a14 string COMMENT 'a14列注释',

a15 string COMMENT 'a15列注释',

a16 string COMMENT 'a16列注释',

a17 string COMMENT 'a17列注释',

a18 string COMMENT 'a18列注释',

a19 string COMMENT 'a19列注释',

a20 string COMMENT 'a20列注释',

a21 string COMMENT 'a21列注释',

a22 string COMMENT 'a22列注释',

a23 string COMMENT 'a23列注释',

a24 string COMMENT 'a24列注释',

a25 string COMMENT 'a25列注释',

a26 string COMMENT 'a26列注释',

a27 string COMMENT 'a27列注释')

COMMENT 'GN数据测试表'

PARTITIONED BY (MONTH\_PART STRING COMMENT '月份分区',

DAY\_PART STRING COMMENT '日期分区')

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001'

STORED AS RCFILE;

### 日期分区表

**示例：**

CREATE TABLE ODS.ODS\_GN\_TABLE(

a1 string COMMENT 'a1列注释',

a2 string COMMENT 'a2列注释',

a3 string COMMENT 'a3列注释',

a4 string COMMENT 'a4列注释',

a5 string COMMENT 'a5列注释',

a6 string COMMENT 'a6列注释',

a7 string COMMENT 'a7列注释',

a8 string COMMENT 'a8列注释',

a9 string COMMENT 'a9列注释',

a10 string COMMENT 'a10列注释',

a11 string COMMENT 'a11列注释',

a12 string COMMENT 'a12列注释',

a13 string COMMENT 'a13列注释',

a14 string COMMENT 'a14列注释',

a15 string COMMENT 'a15列注释',

a16 string COMMENT 'a16列注释',

a17 string COMMENT 'a17列注释',

a18 string COMMENT 'a18列注释',

a19 string COMMENT 'a19列注释',

a20 string COMMENT 'a20列注释',

a21 string COMMENT 'a21列注释',

a22 string COMMENT 'a22列注释',

a23 string COMMENT 'a23列注释',

a24 string COMMENT 'a24列注释',

a25 string COMMENT 'a25列注释',

a26 string COMMENT 'a26列注释',

a27 string COMMENT 'a27列注释')

COMMENT 'GN数据测试表'

PARTITIONED BY (DATE\_PART STRING COMMENT '日期分区')

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001'

STORED AS RCFILE;

## Hive表数据在HDFS存放位置

表数据文件存放于hdfs上，目录为：/user/hive/warehouse/模式名.db/表名/分区名/[子分区名]。

**示例：**/user/hive/warehouse/src.db|ods.db|dwd.db|dwa.db|dwf.db/dwa\_s\_d\_use\_mb\_voice\_comm\_d/month\_part=201405/day\_part=31

# Hive脚本开发规范

## 目录规划

Hive客户端定义Hive脚本存放目录（hive\_sql），hive\_sql下分为src、ods、dwd、dwa四个目录，每个目录下再分为business\_sql、config、createtable、logs。

business\_sql：存放业务逻辑脚本

config：存放配置文件

createtable：存放建表语句

logs：存放Hive脚本执行日志

**目录树示例如下：**

/shbigdata/src/business\_sql

config

createtable

logs

/shbigdata /ods/business\_sql

config

createtable

logs

/shbigdata /dwd/business\_sql

config

createtable

logs

/shbigdata/dwa/business\_sql

config

createtable

logs

## 数据加载

### 配置文件

**模板示例内容如下：**

#!/bin/bash

#定义Hive数据导出配置信息，主要设备公共环境变量、字符集、目录、数据库连接字符串等

#定义HIVE脚本执行日志存放目录

log\_file\_directory=/shbigdata/src/logs/

#定义oracle客户端环境变量

oracle\_sid=ODSDB11

oracle\_base=/u01/app/oracle

oracle\_home=/u01/app/oracle/product/11.2.0/dbhome\_1

#定义ORACLE日志过程的名称、所属用户，以及TNS名称

ora\_log\_user=dw

ora\_log\_user\_passwd=dw123

ora\_log\_tns=bonc

insert\_log\_prc=p\_insert\_log

update\_log\_prc=p\_update\_log

charset=UTF8

#定义业务脚本存放目录

business\_sql\_directory=/shbigdata/src/business\_sql/

#定义数据文件存放目录

data\_file\_directory=/data/disk5/sh\_test\_data/

### 加载脚本

脚本名称：/shbigdata/src/business\_sql/src\_load\_data.sh

使用方法：sh /shbigdata/src/business\_sql/src\_load\_data.sh pxg\_test pxg\*.txt 20150718

参数说明：

参数1：表名

参数2：文件名

参数3：账期

**模板示例内容如下：**

#! /bin/bash

###################################################################

#\*名称 --%@NAME:SRC数据加载

#\*功能描述 --%@COMMENT:SRC数据加载

#\*执行周期 --%@PERIOD:日

#\*参数 --%@PARAM:V\_DATE 帐期 YYYYMMDD

#\*创建人 --%@CREATOR: 创建者

#\*创建时间 --%@CREATED\_TIME: 2015-05-25

#\*迁移人：

#\*迁移时间：

#\*层次---%@LEVEL:ODS层

#\*主题域---%@MASTER\_FIELD:客户域

#\*备注 --%@REMARK:

#\*修改记录 --%@MODIFY:

#\*所属于实体--%@ENTITY:

#\*来源表 --%@FROM:FILE

#\*目标表 --%@TO:SRC.GN\_TABLE

####################################################################

#读取日期参数

table\_name=$1

data\_file=$2

v\_date=$3

#指定此脚本名称，此变量很重要，请必须正确填写!!!!!

typeset -u v\_procname

v\_procname='p\_'$table\_name

#指定此脚本所属数据库名称(等同于oracle中用户名)，此变量很重要，请必须正确填写!!!!!

typeset -u v\_pkg

v\_pkg='SRC'

#定义记录数据记录条数的变量

rowline=0

#切分数据账期，此变量很重要，请必须正确填写!!!!!

v\_month=`echo $v\_date |cut -c 1-6`

v\_day=`echo $v\_date |cut -c 7-8`

#################读取配置文件信息####################

#指定配置文件存放位置，此变量非常重要，请必须填写!!!!!!

config\_file=/shbigdata/src/config/execshell.config

#读取HIVE脚本执行日志存放目录

log\_file\_directory=`awk -F= '{if($1 =="log\_file\_directory") print $2}' $config\_file`

#指定输出日志目录，用于记录hive脚本执行信息

v\_logfile=$log\_file\_directory$v\_procname'.log'

#读取ORACLE客户端环境变量，并加载，此处export后边的ORACLE\_SID、ORACLE\_BASE、ORACLE\_HOME、PATH、LD\_LIBRARY\_PATH需要大写

oracle\_sid=`awk -F= '{if($1 =="oracle\_sid") print $2}' $config\_file`

export ORACLE\_SID=$oracle\_sid

oracle\_base=`awk -F= '{if($1 =="oracle\_base") print $2}' $config\_file`

export ORACLE\_BASE=$oracle\_base

oracle\_home=`awk -F= '{if($1 =="oracle\_home") print $2}' $config\_file`

export ORACLE\_HOME=$oracle\_home

export PATH=$PATH:$HOME/bin:$oracle\_home/bin

export LD\_LIBRARY\_PATH=$oracle\_home/lib:/usr/lib

#读取ORACLE日志过程的名称、所属用户，以及TNS名称

ora\_log\_user=`awk -F= '{if($1 =="ora\_log\_user") print $2}' $config\_file`

ora\_log\_user\_passwd=`awk -F= '{if($1 =="ora\_log\_user\_passwd") print $2}' $config\_file`

ora\_log\_tns=`awk -F= '{if($1 =="ora\_log\_tns") print $2}' $config\_file`

insert\_log\_prc=`awk -F= '{if($1 =="insert\_log\_prc") print $2}' $config\_file`

update\_log\_prc=`awk -F= '{if($1 =="update\_log\_prc") print $2}' $config\_file`

#执行日志插入过程

v\_insert\_log\_sql="EXECUTE DECLARE BEGIN $ora\_log\_user.$insert\_log\_prc('$v\_date','$v\_pkg','$v\_procname','12',sysdate); END;"

pram\_database\_config=`sqlplus -S $ora\_log\_user/$ora\_log\_user\_passwd@$ora\_log\_tns << !

$v\_insert\_log\_sql

exit

!`

#插入数据,此处代码为业务逻辑，需要根据实际情况修改!!!!!!!!

#读取数据文件存放目录

data\_file\_directory=`awk -F= '{if($1 =="data\_file\_directory") print $2}' $config\_file`

#此处需要指定对应SQL脚本中的输入参数，非常重要，必须准确填写!!!!!!!

load\_data\_file=$data\_file\_directory'day\_'$v\_date'/'$data\_file

v\_sql="load data local inpath '$load\_data\_file' overwrite into table $table\_name"

#Hive调用业务逻辑脚本，进行数据加工作业

hive -e "

use $v\_pkg;

$v\_sql;

" 2>&1 |tee $v\_logfile >>/dev/null

#获取sql执行结果信息

v\_result=`cat $v\_logfile | grep -s "FAILED" | awk -F ":" '{print $1}'` >>/dev/null

#判断数据是否导出成功

if [ "$v\_result" != "FAILED" ]; then

v\_retcode='SUCCESS'

v\_retinfo='结束'

else

v\_retcode='FAIL'

v\_retinfo=`cat $v\_logfile | grep -s "FAILED"` >>/dev/null

fi

#统计原数据库中记录数，此处需要指定数据库名、表名、查询的分区，此处必须正确修改!!!!!!!!

rowline=`hive -e " select count(1) coun from src.$table\_name "

exit

!`

#对ORACLE日志表进行日志更新操作

v\_update\_log\_sql="EXECUTE DECLARE BEGIN $ora\_log\_user.$update\_log\_prc('$v\_date','$v\_pkg','$v\_procname','$v\_retcode','$v\_retinfo',sysdate,$rowline); END;"

#读取ORACLE数据库字符集

charset=`awk -F= '{if($1 =="charset") print $2}' $config\_file`

export NLS\_LANG="AMERICAN\_AMERICA."$charset

pram\_database\_config=`sqlplus -S $ora\_log\_user/$ora\_log\_user\_passwd@$ora\_log\_tns << !

$v\_update\_log\_sql

exit

!`

## 加工脚本模板

此处以ODS层某一业务处理逻辑为范本，进行脚本模板定义。

### 配置文件

存放于/shbigdata/ods/config/，配置文件统一命名为execshell.config。

**配置文件示例内容如下：**

#!/bin/bash

#定义Hive数据导出配置信息，主要设备公共环境变量、字符集、目录、数据库连接字符串等

**#定义HIVE脚本执行日志存放目录**

log\_file\_directory=/shbigdata/ods/logs/

#定义oracle客户端环境变量

oracle\_sid=oracle

oracle\_base=/u01/app/oracle

oracle\_home=/u01/app/oracle/product/11.2.0/dbhome\_1

#定义ORACLE日志过程的名称、所属用户，以及TNS名称

ora\_log\_user=dw

ora\_log\_user\_passwd=dw

ora\_log\_tns=oracle

insert\_log\_prc=p\_insert\_log

update\_log\_prc=p\_update\_log

charset=UTF8

**#定义业务脚本存放目录**

business\_sql\_directory=/shbigdata/ods/business\_sql/

### 业务逻辑脚本

#### 基础逻辑

基础逻辑脚本，包含了日志记录、注释、配置文件读取等工作，并调用业务逻辑脚本。

存放于../hive\_sql/ods/business\_sql/，脚本命名以**P\_数据表名.sh，定义一个输入参数，输入参数为业务数据账期**

**模板示例内容如下：**

#! /bin/bash

###################################################################

#\*名称 --%@NAME:GN详单

#\*功能描述 --%@COMMENT: GN详单

#\*执行周期 --%@PERIOD:日

#\*参数 --%@PARAM:V\_DATE 帐期 YYYYMMDD

#\*参数 --%@PARAM:V\_MONTH 月份 YYYYMM

#\*参数 --%@PARAM:V\_DAY 当天 DD

#\*创建人 --%@CREATOR: 陈强

#\*创建时间 --%@CREATED\_TIME: 2010-02-25

#\*层次---%@LEVEL:ODS层

#\*主题域---%@MASTER\_FIELD:客户域

#\*备注 --%@REMARK:

#\*修改记录 --%@MODIFY:

#\*所属于实体--%@ENTITY:

#\*来源表 --%@FROM:SRC.SRC\_GN\_TABLE

#\*目标表 --%@TO:ODS.ODS\_GN\_TABLE

####################################################################

#读取日期参数

v\_date=$1

**#指定此脚本目标数据表，此变量很重要，请必须正确填写!!!!!**

**table\_name='ods\_gn\_table'**

#**指定此脚本名称，此变量很重要，请必须正确填写!!!!!**

typeset -u v\_procname

**v\_procname='p\_ '$ table\_name**

**#指定此脚本所属数据库名称(等同于oracle中用户名)，此变量很重要，请必须正确填写!!!!!**

typeset -u v\_pkg**='ods'**

#定义记录数据记录条数的变量

rowline=0

**#切分数据账期，此变量很重要，请必须正确填写!!!!!**

**v\_month=`echo $v\_date |cut -c 1-6`**

**v\_day=`echo $v\_date |cut -c 7-8`**

#################读取配置文件信息####################

**#指定配置文件存放位置，此变量非常重要，请必须填写!!!!!!**

typeset -l v\_pkg=$v\_pkg

**config\_file="/shbigdata/"$v\_pkg"/config/execshell.config"**

typeset -u v\_pkg=$v\_pkg

#读取HIVE脚本执行日志存放目录

log\_file\_directory=`awk -F= '{if($1 =="log\_file\_directory") print $2}' $config\_file`

**#指定输出日志目录，用于记录hive脚本执行信息**

v\_logfile=$log\_file\_directory$v\_procname'.log'

#读取ORACLE客户端环境变量，并加载，此处export后边的ORACLE\_SID、ORACLE\_BASE、ORACLE\_HOME、PATH、LD\_LIBRARY\_PATH需要大写

oracle\_sid=`awk -F= '{if($1 =="oracle\_sid") print $2}' $config\_file`

export ORACLE\_SID=$oracle\_sid

oracle\_base=`awk -F= '{if($1 =="oracle\_base") print $2}' $config\_file`

export ORACLE\_BASE=$oracle\_base

oracle\_home=`awk -F= '{if($1 =="oracle\_home") print $2}' $config\_file`

export ORACLE\_HOME=$oracle\_home

export PATH=$PATH:$HOME/bin:$oracle\_home/bin

export LD\_LIBRARY\_PATH=$oracle\_home/lib:/usr/lib

#读取ORACLE日志过程的名称、所属用户，以及TNS名称

ora\_log\_user=`awk -F= '{if($1 =="ora\_log\_user") print $2}' $config\_file`

ora\_log\_user\_passwd=`awk -F= '{if($1 =="ora\_log\_user\_passwd") print $2}' $config\_file`

ora\_log\_tns=`awk -F= '{if($1 =="ora\_log\_tns") print $2}' $config\_file`

insert\_log\_prc=`awk -F= '{if($1 =="insert\_log\_prc") print $2}' $config\_file`

update\_log\_prc=`awk -F= '{if($1 =="update\_log\_prc") print $2}' $config\_file`

#执行日志插入过程

v\_insert\_log\_sql="EXECUTE DECLARE BEGIN $ora\_log\_user.$insert\_log\_prc('$v\_date','$v\_pkg','$v\_procname','12',sysdate); END;"

pram\_database\_config=`sqlplus -S $ora\_log\_user/$ora\_log\_user\_passwd@$ora\_log\_tns << !

$v\_insert\_log\_sql

exit

!`

#插入数据,此处代码为业务逻辑，需要根据实际情况修改!!!!!!!!

#读取业务脚本存放目录

business\_sql\_directory=`awk -F= '{if($1 =="business\_sql\_directory") print $2}' $config\_file`

**#此处需要指定对应SQL脚本中的输入参数，非常重要，必须准确填写!!!!!!!**

v\_sql=`sh $business\_sql\_directory$v\_procname'\_SQL.sh' $v\_month $v\_day`

**#Hive调用业务逻辑脚本，进行数据加工作业**

hive -e "

use $v\_pkg;

$v\_sql

" 2>&1 |tee $v\_logfile >>/dev/null

#获取sql执行结果信息

v\_result=`cat $v\_logfile | grep -s "FAILED" | awk -F ":" '{print $1}'` >>/dev/null

#判断数据是否导出成功

if [ "$v\_result" != "FAILED" ]; then

v\_retcode='SUCCESS'

v\_retinfo='结束'

else

v\_retcode='FAIL'

v\_retinfo=`cat $v\_logfile | grep -s "FAILED"` >>/dev/null

fi

**#统计原数据库中记录数，此处需要指定数据库名、表名、查询的分区，此处必须正确修改!!!!!!!!**

**v\_sql="select count(1) coun from $v\_pkg.$table\_name where month\_part = '$v\_month' and day\_part = '$v\_day'"**

rowline=`hive -e "$v\_sql"

exit

!`

#对ORACLE日志表进行日志更新操作

v\_update\_log\_sql="EXECUTE DECLARE BEGIN $ora\_log\_user.$update\_log\_prc('$v\_date','$v\_pkg','$v\_procname','$v\_retcode','$v\_retinfo',sysdate,$rowline); END;"

#读取ORACLE数据库字符集

charset=`awk -F= '{if($1 =="charset") print $2}' $config\_file`

export NLS\_LANG="AMERICAN\_AMERICA."$charset

pram\_database\_config=`sqlplus -S $ora\_log\_user/$ora\_log\_user\_passwd@$ora\_log\_tns << !

$v\_update\_log\_sql

exit

!`

#### 业务逻辑

**模板示例内容如下：**

#/bin/bash

**#业务逻辑输入参数，此处很重要，需要根据实际业务情况认真设定！！！！**

**v\_month=$1**

**v\_day=$2**

echo "

insert overwrite table ods.ods\_gn\_table partition (month\_part = '$v\_month',day\_part = '$v\_day')

select

a1 ,

a2 ,

a3 ,

a4 ,

a5 ,

a6 ,

a7 ,

a8 ,

a9 ,

a10,

a11,

a12,

a13,

a14,

a15,

a16,

a17,

a18,

a19,

a20,

a21,

a22,

a23,

a24,

a25,

a26,

a27

from src.gn\_table;"

# Hive UDF自定义函数开发规范

## 开发步骤

UDF简称自定义函数，它是Hive函数库的扩展，自定义函数UDF在MapReduce执行阶段发挥作用。开发步骤如下：

1. 给hive.ql.exec.UDF包开发一个自定义函数类，从UDF继承。自定义函数类实现evaluate方法。
2. 在FunctionRegistry类中注册开发的自定义函数类。
3. 打包发布至Hive客户端。

## 开发工具

Eclipse是一款开源的、基于Java的可扩展开发平台。Hadoop开发人员可通过在Eclipse上面开发UDF。

## UDF函数示例

**1）开发UDF函数类**

文件名及路径：/hive-0.12.0/src/ql/src/java/org/apache/hadoop/hive/ql/udf/UDFHelloWorld.java

**package** org.apache.hadoop.hive.ql.udf;

**import** org.apache.hadoop.hive.ql.exec.UDF;

**import** org.apache.hadoop.io.Text;

**public** **class** UDFHelloWorld **extends** UDF {

**public** String evaluate(String str) {

**if** (str == **null**) {

**return** **null**;

}

**return** "HelloWorld " + str;

}

**public** **static** **void** main(String[] args) {

helloUDF uf = **new** helloUDF();

//Text t = new Text("gfsg");

System.*out*.println(uf.evaluate("nihao").toString());

}

}

**2）UDF类注册，注册方法**

文件名及路径：/hive-0.12.0/src/ql/src/java/org/apache/hadoop/hive/ql/exec/FunctionRegistry.java

package org.apache.hadoop.hive.ql.exec;

import org.apache.hadoop.hive.ql.udf.UDFHelloWorld;

/\*\*

\* FunctionRegistry.

\*/

public final class FunctionRegistry {

static {

registerGenericUDF("concat", GenericUDFConcat.class);

registerUDF("substr", UDFSubstr.class, false);

registerUDF("substring", UDFSubstr.class, false);

registerUDF("space", UDFSpace.class, false);

registerUDF("repeat", UDFRepeat.class, false);

registerUDF("ascii", UDFAscii.class, false);

registerUDF("lpad", UDFLpad.class, false);

registerUDF("rpad", UDFRpad.class, false);

registerUDF("Hello", UDFHelloWorld.class, false);

registerGenericUDF("size", GenericUDFSize.class);

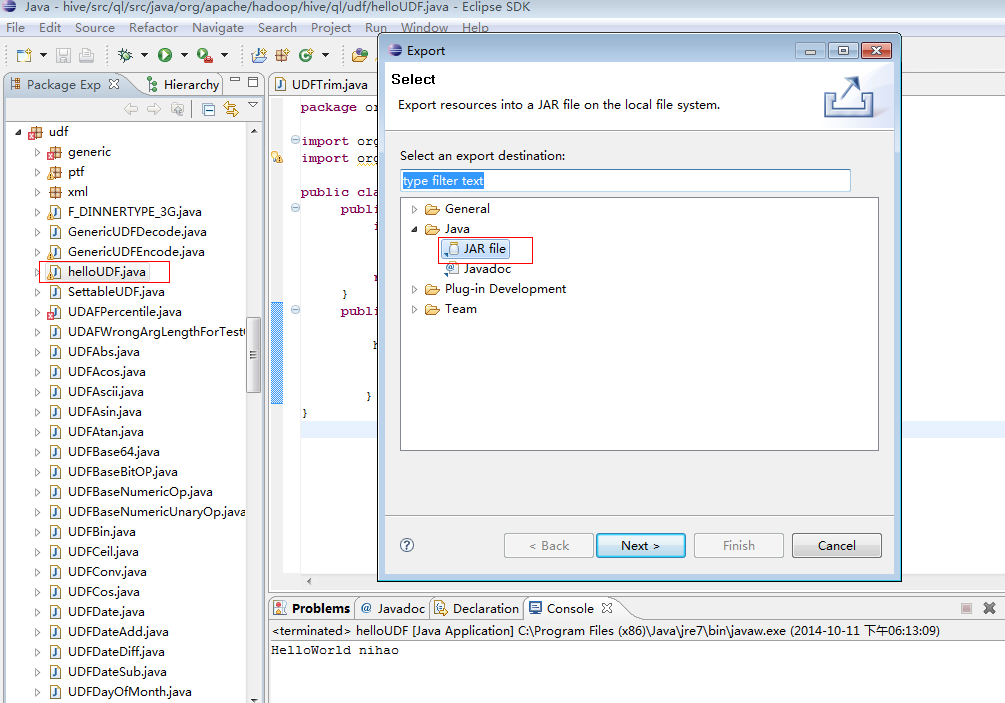
……………………………………………………

……………………………………………………

**3）Jar包发布路径**

发布路径：/opt/boh/hive/lib/hive-exec-0.12.0-cdh5.0.0.jar

上传至hadoop集群执行脚本的hive客户端。



## Hive UDF函数(逐步补充)

### UDF函数列表

函数清单及其功能

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### UDF函数说明

* TO\_DATE函数

Select to\_date('20140909111111','YYYYMMDDHH24miss') from test;

返回结果：2014-09-09 11:11:11

* ADD\_MONTHS函数

select add\_months(to\_date('20140909111111','YYYYMMDDHH24miss'),1) from test;

返回结果：2014-10-09 11:11:11

* date\_tostring函数

select date\_tostring(to\_date('20140909111111','YYYYMMDDHH24miss'),'YYYY-MM-DD') from test;

返回结果：2014-09-09

* MONTHS\_BETWEEN函数

select MONTHS\_BETWEEN(to\_date('20140909111111','YYYYMMDDHH24miss'),to\_date('20140706111111','YYYYMMDDHH24miss')) from test;

返回结果：2.096774193548387

* f\_age函数

select f\_age('511024198710148199') from test;

返回结果：127

* f\_checkidcard函数

select f\_checkidcard('511024198710148199') from test;

返回结果：1

# 附录一：Hive基本语法

## 参考文档1

Hive 的官方文档中对查询语言有了很详细的描述，请参考：http://wiki.apache.org/hadoop/Hive/LanguageManual。

### Create Table

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table\_name

[(col\_name data\_type [COMMENT col\_comment], ...)]

[COMMENT table\_comment]

[PARTITIONED BY (col\_name data\_type

[COMMENT col\_comment], ...)]

[CLUSTERED BY (col\_name, col\_name, ...)

[SORTED BY (col\_name [ASC|DESC], ...)]

INTO num\_buckets BUCKETS]

[ROW FORMAT row\_format]

[STORED AS file\_format]

[LOCATION hdfs\_path]

CREATE TABLE 创建一个指定名字的表。如果相同名字的表已经存在，则抛出异常；用户可以用 IF NOT EXIST 选项来忽略这个异常。

EXTERNAL 关键字可以让用户创建一个外部表，在建表的同时指定一个指向实际数据的路径（LOCATION），Hive 创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径，不对数据的位置做任何改变。在删除表的时候，内部表的元数据和数据会被一起删除，而外部表只删除元数据，不删除数据。

LIKE 允许用户复制现有的表结构，但是不复制数据。

用户在建表的时候可以自定义 SerDe 或者使用自带的 SerDe。如果没有指定 ROW FORMAT 或者 ROW FORMAT DELIMITED，将会使用自带的 SerDe。在建表的时候，用户还需要为表指定列，用户在指定表的列的同时也会指定自定义的 SerDe，Hive 通过 SerDe 确定表的具体的列的数据。

如果文件数据是纯文本，可以使用 STORED AS TEXTFILE。如果数据需要压缩，使用 STORED AS SEQUENCE 。

有分区的表可以在创建的时候使用 PARTITIONED BY 语句。一个表可以拥有一个或者多个分区，每一个分区单独存在一个目录下。而且，表和分区都可以对某个列进行 CLUSTERED BY 操作，将若干个列放入一个桶（bucket）中。也可以利用SORT BY 对数据进行排序。这样可以为特定应用提高性能。

表名和列名不区分大小写，SerDe 和属性名区分大小写。表和列的注释是字符串。

### Drop Table

删除一个内部表的同时会同时删除表的元数据和数据。删除一个外部表，只删除元数据而保留数据。

### Alter Table

Alter table 语句允许用户改变现有表的结构。用户可以增加列/分区，改变serde，增加表和 serde 熟悉，表本身重命名。

**Add Partitions**

ALTER TABLE table\_name ADD

partition\_spec [ LOCATION 'location1' ]

partition\_spec [ LOCATION 'location2' ] ...

partition\_spec:

: PARTITION (partition\_col = partition\_col\_value,

partition\_col = partiton\_col\_value, ...)

用户可以用 ALTER TABLE ADD PARTITION 来向一个表中增加分区。当分区名是字符串时加引号。

ALTER TABLE page\_view ADD

PARTITION (dt='2008-08-08', country='us')

location '/path/to/us/part080808'

PARTITION (dt='2008-08-09', country='us')

location '/path/to/us/part080809';

**DROP PARTITION**

ALTER TABLE table\_name DROP

partition\_spec, partition\_spec,...

用户可以用 ALTER TABLE DROP PARTITION 来删除分区。分区的元数据和数据将被一并删除。

ALTER TABLE page\_view

DROP PARTITION (dt='2008-08-08', country='us');

**RENAME TABLE**

ALTER TABLE table\_name RENAME TO new\_table\_name

这个命令可以让用户为表更名。数据所在的位置和分区名并不改变。换而言之，老的表名并未“释放”，对老表的更改会改变新表的数据。

**Change Column Name/Type/Position/Comment**

ALTER TABLE table\_name CHANGE [COLUMN]

col\_old\_name col\_new\_name column\_type

[COMMENT col\_comment]

[FIRST|AFTER column\_name]

这个命令可以允许用户修改一个列的名称、数据类型、注释或者位置。

比如：

CREATE TABLE test\_change (a int, b int, c int);

ALTER TABLE test\_change CHANGE a a1 INT; 将 a 列的名字改为 a1.

ALTER TABLE test\_change CHANGE a a1 STRING AFTER b; 将 a 列的名字改为 a1，a 列的数据类型改为 string，并将它放置在列 b 之后。新的表结构为： b int, a1 string, c int.

ALTER TABLE test\_change CHANGE b b1 INT FIRST; 会将 b 列的名字修改为 b1, 并将它放在第一列。新表的结构为： b1 int, a string, c int.

注意：对列的改变只会修改 Hive 的元数据，而不会改变实际数据。用户应该确定保证元数据定义和实际数据结构的一致性。

**Add/Replace Columns**

ALTER TABLE table\_name ADD|REPLACE

COLUMNS (col\_name data\_type [COMMENT col\_comment], ...)

ADD COLUMNS 允许用户在当前列的末尾增加新的列，但是在分区列之前。

REPLACE COLUMNS 删除以后的列，加入新的列。只有在使用 native 的 SerDE（DynamicSerDe or MetadataTypeColumnsetSerDe）的时候才可以这么做。

**Alter Table Properties**

ALTER TABLE table\_name SET TBLPROPERTIES table\_properties

table\_properties:

: (property\_name = property\_value, property\_name = property\_value, ... )

用户可以用这个命令向表中增加 metadata，目前 last\_modified\_user，last\_modified\_time 属性都是由 Hive 自动管理的。用户可以向列表中增加自己的属性。可以使用 DESCRIBE EXTENDED TABLE 来获得这些信息。

**Add Serde Properties**

ALTER TABLE table\_name

SET SERDE serde\_class\_name

[WITH SERDEPROPERTIES serde\_properties]

ALTER TABLE table\_name

SET SERDEPROPERTIES serde\_properties

serde\_properties:

: (property\_name = property\_value,

property\_name = property\_value, ... )

这个命令允许用户向 SerDe 对象增加用户定义的元数据。Hive 为了序列化和反序列化数据，将会初始化 SerDe 属性，并将属性传给表的 SerDe。如此，用户可以为自定义的 SerDe 存储属性。

**Alter Table File Format and Organization**

ALTER TABLE table\_name SET FILEFORMAT file\_format

ALTER TABLE table\_name CLUSTERED BY (col\_name, col\_name, ...)

[SORTED BY (col\_name, ...)] INTO num\_buckets BUCKETS

这个命令修改了表的物理存储属性。

### Loading files into table

当数据被加载至表中时，不会对数据进行任何转换。Load 操作只是将数据复制/移动至 Hive 表对应的位置。

**Syntax:**

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE]

INTO TABLE tablename

[PARTITION (partcol1=val1, partcol2=val2 ...)]

**Synopsis:**

Load 操作只是单纯的复制/移动操作，将数据文件移动到 Hive 表对应的位置。

* filepath 可以是：
  + 相对路径，例如：project/data1
  + 绝对路径，例如： /user/hive/project/data1
  + 包含模式的完整 URI，例如：hdfs://namenode:9000/user/hive/project/data1
* 加载的目标可以是一个表或者分区。如果表包含分区，必须指定每一个分区的分区名。
* filepath 可以引用一个文件（这种情况下，Hive 会将文件移动到表所对应的目录中）或者是一个目录（在这种情况下，Hive 会将目录中的所有文件移动至表所对应的目录中）。
* 如果指定了 LOCAL，那么：
  + load 命令会去查找本地文件系统中的 filepath。如果发现是相对路径，则路径会被解释为相对于当前用户的当前路径。用户也可以为本地文件指定一个完整的 URI，比如：file:///user/hive/project/data1.
  + load 命令会将 filepath 中的文件复制到目标文件系统中。目标文件系统由表的位置属性决定。被复制的数据文件移动到表的数据对应的位置。
* 如果没有指定 LOCAL 关键字，如果 filepath 指向的是一个完整的 URI，hive 会直接使用这个 URI。 否则：
  + 如果没有指定 schema 或者 authority，Hive 会使用在 hadoop 配置文件中定义的 schema 和 authority，fs.default.name 指定了 Namenode 的 URI。
  + 如果路径不是绝对的，Hive 相对于 /user/ 进行解释。
  + Hive 会将 filepath 中指定的文件内容移动到 table （或者 partition）所指定的路径中。
* 如果使用了 OVERWRITE 关键字，则目标表（或者分区）中的内容（如果有）会被删除，然后再将 filepath 指向的文件/目录中的内容添加到表/分区中。
* 如果目标表（分区）已经有一个文件，并且文件名和 filepath 中的文件名冲突，那么现有的文件会被新文件所替代。

### SELECT

**Syntax**

SELECT [ALL | DISTINCT] select\_expr, select\_expr, ...

FROM table\_reference

[WHERE where\_condition]

[GROUP BY col\_list]

[

CLUSTER BY col\_list

| [DISTRIBUTE BY col\_list]

[SORT BY col\_list]

]

[LIMIT number]

* 一个SELECT语句可以是一个union查询或一个子查询的一部分。
* table\_reference是查询的输入，可以是一个普通表、一个视图、一个join或一个子查询
* 简单查询。例如，下面这一语句从t1表中查询所有列的信息。

SELECT \* FROM t1

**WHERE Clause**

where condition 是一个布尔表达式。例如，下面的查询语句只返回销售记录大于 10，且归属地属于美国的销售代表。Hive 不支持在WHERE 子句中的 IN，EXIST 或子查询。

SELECT \* FROM sales WHERE amount > 10 AND region = "US"

**ALL and DISTINCT Clauses**

使用ALL和DISTINCT选项区分对重复记录的处理。默认是ALL，表示查询所有记录。DISTINCT表示去掉重复的记录。

hive> SELECT col1, col2 FROM t1

1 3

1 3

1 4

2 5

hive> SELECT DISTINCT col1, col2 FROM t1

1 3

1 4

2 5

hive> SELECT DISTINCT col1 FROM t1

1

2

**基于Partition的查询**

一般 SELECT 查询会扫描整个表（除非是为了抽样查询）。但是如果一个表使用 PARTITIONED BY 子句建表，查询就可以利用分区剪枝（input pruning）的特性，只扫描一个表中它关心的那一部分。Hive 当前的实现是，只有分区断言出现在离 FROM 子句最近的那个WHERE 子句中，才会启用分区剪枝。例如，如果 page\_views 表使用 date 列分区，以下语句只会读取分区为‘2008-03-01’的数据。

SELECT page\_views.\*

FROM page\_views

WHERE page\_views.date >= '2008-03-01'

AND page\_views.date <= '2008-03-31';

**HAVING Clause**

Hive 现在不支持 HAVING 子句。可以将 HAVING 子句转化为一个字查询，例如：

SELECT col1 FROM t1 GROUP BY col1 HAVING SUM(col2) > 10

可以用以下查询来表达：

SELECT col1 FROM (SELECT col1, SUM(col2) AS col2sum

FROM t1 GROUP BY col1) t2

WHERE t2.col2sum > 10

**LIMIT Clause**

Limit 可以限制查询的记录数。查询的结果是随机选择的。下面的查询语句从 t1 表中随机查询5条记录：

SELECT \* FROM t1 LIMIT 5

Top k 查询。下面的查询语句查询销售记录最大的 5 个销售代表。

SET mapred.reduce.tasks = 1

SELECT \* FROM sales SORT BY amount DESC LIMIT 5

**REGEX Column Specification**

SELECT 语句可以使用正则表达式做列选择，下面的语句查询除了 ds 和 hr 之外的所有列：

SELECT `(ds|hr)?+.+` FROM sales

### Join

**Syntax**

join\_table:

table\_reference JOIN table\_factor [join\_condition]

| table\_reference {LEFT|RIGHT|FULL} [OUTER]

JOIN table\_reference join\_condition

| table\_reference LEFT SEMI 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\_r( AND equality\_expression )\*

equality\_expression:

expression = expression

Hive 只支持等值连接（equality joins）、外连接（outer joins）和（left semi joins???）。Hive 不支持所有非等值的连接，因为非等值连接非常难转化到 map/reduce 任务。另外，Hive 支持多于 2 个表的连接。

写 join 查询时，需要注意几个关键点：  
1. 只支持等值join，例如：

SELECT a.\* FROM a JOIN b ON (a.id = b.id)

SELECT a.\* FROM a JOIN b

ON (a.id = b.id AND a.department = b.department)

是正确的，然而:

SELECT a.\* FROM a JOIN b ON (a.id b.id)

是错误的。

2. 可以 join 多于 2 个表，例如

SELECT a.val, b.val, c.val FROM a JOIN b

ON (a.key = b.key1) JOIN c ON (c.key = b.key2)

如果join中多个表的 join key 是同一个，则 join 会被转化为单个 map/reduce 任务，例如：

SELECT a.val, b.val, c.val FROM a JOIN b

ON (a.key = b.key1) JOIN c

ON (c.key = b.key1)

被转化为单个 map/reduce 任务，因为 join 中只使用了 b.key1 作为 join key。

SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1)

JOIN c ON (c.key = b.key2)

而这一 join 被转化为 2 个 map/reduce 任务。因为 b.key1 用于第一次 join 条件，而 b.key2 用于第二次 join。

join 时，每次 map/reduce 任务的逻辑是这样的：reducer 会缓存 join 序列中除了最后一个表的所有表的记录，再通过最后一个表将结果序列化到文件系统。这一实现有助于在 reduce 端减少内存的使用量。实践中，应该把最大的那个表写在最后（否则会因为缓存浪费大量内存）。例如：

SELECT a.val, b.val, c.val FROM a

JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1)

所有表都使用同一个 join key（使用 1 次 map/reduce 任务计算）。Reduce 端会缓存 a 表和 b 表的记录，然后每次取得一个 c 表的记录就计算一次 join 结果，类似的还有：

SELECT a.val, b.val, c.val FROM a

JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2)

这里用了 2 次 map/reduce 任务。第一次缓存 a 表，用 b 表序列化；第二次缓存第一次 map/reduce 任务的结果，然后用 c 表序列化。

LEFT，RIGHT 和 FULL OUTER 关键字用于处理 join 中空记录的情况，例如：

SELECT a.val, b.val FROM a LEFT OUTER

JOIN b ON (a.key=b.key)

对应所有 a 表中的记录都有一条记录输出。输出的结果应该是 a.val, b.val，当 a.key=b.key 时，而当 b.key 中找不到等值的 a.key 记录时也会输出 a.val, NULL。“FROM a LEFT OUTER JOIN b”这句一定要写在同一行——意思是 a 表在 b 表的**左边**，所以 a 表中的所有记录都被保留了；“a RIGHT OUTER JOIN b”会保留所有 b 表的记录。OUTER JOIN 语义应该是遵循标准 SQL spec的。

Join 发生在 WHERE 子句**之前**。如果你想限制 join 的输出，应该在 WHERE 子句中写过滤条件——或是在 join 子句中写。这里面一个容易混淆的问题是表分区的情况：

SELECT a.val, b.val FROM a

LEFT OUTER JOIN b ON (a.key=b.key)

WHERE a.ds='2009-07-07' AND b.ds='2009-07-07'

会 join a 表到 b 表（OUTER JOIN），列出 a.val 和 b.val 的记录。WHERE 从句中可以使用其他列作为过滤条件。但是，如前所述，如果 b 表中找不到对应 a 表的记录，b 表的所有列都会列出 NULL，**包括 ds 列**。也就是说，join 会过滤 b 表中不能找到匹配 a 表 join key 的所有记录。这样的话，LEFT OUTER 就使得查询结果与 WHERE 子句无关了。解决的办法是在 OUTER JOIN 时使用以下语法：

SELECT a.val, b.val FROM a LEFT OUTER JOIN b

ON (a.key=b.key AND

b.ds='2009-07-07' AND

a.ds='2009-07-07')

这一查询的结果是预先在 join 阶段过滤过的，所以不会存在上述问题。这一逻辑也可以应用于 RIGHT 和 FULL 类型的 join 中。

Join 是不能交换位置的。无论是 LEFT 还是 RIGHT join，都是左连接的。

SELECT a.val1, a.val2, b.val, c.val

FROM a

JOIN b ON (a.key = b.key)

LEFT OUTER JOIN c ON (a.key = c.key)

先 join a 表到 b 表，丢弃掉所有 join key 中不匹配的记录，然后用这一中间结果和 c 表做 join。这一表述有一个不太明显的问题，就是当一个 key 在 a 表和 c 表都存在，但是 b 表中不存在的时候：整个记录在第一次 join，即 a JOIN b 的时候都被丢掉了（包括a.val1，a.val2和a.key），然后我们再和 c 表 join 的时候，如果 c.key 与 a.key 或 b.key 相等，就会得到这样的结果：NULL, NULL, NULL, c.val。

LEFT SEMI JOIN 是 IN/EXISTS 子查询的一种更高效的实现。Hive 当前没有实现 IN/EXISTS 子查询，所以你可以用 LEFT SEMI JOIN 重写你的子查询语句。LEFT SEMI JOIN 的限制是， JOIN 子句中右边的表只能在 ON 子句中设置过滤条件，在 WHERE 子句、SELECT 子句或其他地方过滤都不行。

SELECT a.key, a.value

FROM a

WHERE a.key in

(SELECT b.key

FROM B);

可以被重写为：

SELECT a.key, a.val

FROM a LEFT SEMI JOIN b on (a.key = b.key)

## 参考文档2

Hive 是基于Hadoop 构建的一套数据仓库分析系统，它提供了丰富的SQL查询方式来分析存储在Hadoop 分布式文件系统中的数据，可以将结构

化的数据文件映射为一张数据库表，并提供完整的SQL查询功能，可以将SQL语句转换为MapReduce任务进行运行，通过自己的SQL 去查询分析需

要的内容，这套SQL 简称Hive SQL，使不熟悉mapreduce 的用户很方便的利用SQL 语言查询，汇总，分析数据。而mapreduce开发人员可以把

己写的mapper 和reducer 作为插件来支持Hive 做更复杂的数据分析。

     它与关系型数据库的SQL 略有不同，但支持了绝大多数的语句如DDL、DML 以及常见的聚合函数、连接查询、条件查询。HIVE不适合用于联机

online)事务处理，也不提供实时查询功能。它最适合应用在基于大量不可变数据的批处理作业。  
  
    HIVE的特点：可伸缩（在Hadoop的集群上动态的添加设备），可扩展，容错，输入格式的松散耦合。

     Hive 的官方文档中对查询语言有了很详细的描述，请参考：http://wiki.apache.org/hadoop/Hive/LanguageManual ，本文的内容大部分翻译自该页面，期间加入了一些在使用过程中需要注意到的事项。

### DDL 操作

**DDL**

•建表

•删除表

•修改表结构

•创建／删除视图

•创建数据库

•显示命令

#### 建表：

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table\_name   
  [(col\_name data\_type [COMMENT col\_comment], ...)]   
  [COMMENT table\_comment]   
  [PARTITIONED BY (col\_name data\_type [COMMENT col\_comment], ...)]   
  [CLUSTERED BY (col\_name, col\_name, ...)   
  [SORTED BY (col\_name [ASC|DESC], ...)] INTO num\_buckets BUCKETS]   
  [ROW FORMAT row\_format]   
  [STORED AS file\_format]   
  [LOCATION hdfs\_path]

•CREATE TABLE 创建一个指定名字的表。如果相同名字的表已经存在，则抛出异常；用户可以用 IF NOT EXIST 选项来忽略这个异常

•EXTERNAL 关键字可以让用户创建一个外部表，在建表的同时指定一个指向实际数据的路径（LOCATION）

•LIKE 允许用户复制现有的表结构，但是不复制数据

•COMMENT可以为表与字段增加描述

•ROW FORMAT

    DELIMITED [FIELDS TERMINATED BY char] [COLLECTION ITEMS TERMINATED BY char]

        [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]

   | SERDE serde\_name [WITH SERDEPROPERTIES (property\_name=property\_value, property\_name=property\_value, ...)]

         用户在建表的时候可以自定义 SerDe 或者使用自带的 SerDe。如果没有指定 ROW FORMAT 或者 ROW FORMAT DELIMITED，将会使用自带的 SerDe。在建表的时候，用户还需要为表指定列，用户在指定表的列的同时也会指定自定义的 SerDe，Hive 通过 SerDe 确定表的具体的列的数据。

•STORED AS

            SEQUENCEFILE

            | TEXTFILE

            | RCFILE

            | INPUTFORMAT input\_format\_classname OUTPUTFORMAT             output\_format\_classname

       如果文件数据是纯文本，可以使用 STORED AS TEXTFILE。如果数据需要压缩，使用 STORED AS SEQUENCE 。

#### 创建简单表：

hive> CREATE TABLE pokes (foo INT, bar STRING);

#### ****创建外部表：****

CREATE EXTERNAL TABLE page\_view(viewTime INT, userid BIGINT,

     page\_url STRING, referrer\_url STRING,

     ip STRING COMMENT 'IP Address of the User',

     country STRING COMMENT 'country of origination')

 COMMENT 'This is the staging page view table'

 ROW FORMAT DELIMITED FIELDS TERMINATED BY '\054'

 STORED AS TEXTFILE

 LOCATION '<hdfs\_location>';

#### 建分区表

CREATE TABLE par\_table(viewTime INT, userid BIGINT,

     page\_url STRING, referrer\_url STRING,

     ip STRING COMMENT 'IP Address of the User')

 COMMENT 'This is the page view table'

 PARTITIONED BY(date STRING, pos STRING)

ROW FORMAT DELIMITED ‘\t’

   FIELDS TERMINATED BY '\n'

STORED AS SEQUENCEFILE;

#### 建Bucket表

CREATE TABLE par\_table(viewTime INT, userid BIGINT,

     page\_url STRING, referrer\_url STRING,

     ip STRING COMMENT 'IP Address of the User')

 COMMENT 'This is the page view table'

 PARTITIONED BY(date STRING, pos STRING)

 CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS

 ROW FORMAT DELIMITED ‘\t’

   FIELDS TERMINATED BY '\n'

STORED AS SEQUENCEFILE;

#### 创建表并创建索引字段ds

hive> CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);

#### 复制一个空表

CREATE TABLE empty\_key\_value\_store

LIKE key\_value\_store;

**例子**

create table  user\_info (user\_id int, cid string, ckid string, username string)

row format delimited

fields terminated by '\t'

 lines terminated by '\n';

导入数据表的数据格式是：字段之间是tab键分割，行之间是断行。

及要我们的文件内容格式：

100636  100890  c5c86f4cddc15eb7        yyyvybtvt  
100612  100865  97cc70d411c18b6f        gyvcycy  
100078  100087  ecd6026a15ffddf5        qa000100

#### 显示所有表：

hive> SHOW TABLES;

#### 按正条件（正则表达式）显示表，

hive> SHOW TABLES '.\*s';

#### 修改表结构

•增加分区、删除分区

•重命名表

•修改列的名字、类型、位置、注释

•增加/更新列

•增加表的元数据信息

#### 表添加一列 ：

hive> ALTER TABLE pokes ADD COLUMNS (new\_col INT);

#### 添加一列并增加列字段注释

hive> ALTER TABLE invites ADD COLUMNS (new\_col2 INT COMMENT 'a comment');

#### 更改表名：

hive> ALTER TABLE events RENAME TO 3koobecaf;

#### 删除列：

hive> DROP TABLE pokes;

#### 增加、删除分区

•增加

ALTER TABLE table\_name ADD [IF NOT EXISTS] partition\_spec [ LOCATION 'location1' ] partition\_spec [ LOCATION 'location2' ] ...

      partition\_spec:

  : PARTITION (partition\_col = partition\_col\_value, partition\_col = partiton\_col\_value, ...)

•删除

ALTER TABLE table\_name DROP partition\_spec, partition\_spec,...

#### 重命名表

•ALTER TABLE table\_name RENAME TO new\_table\_name

#### 修改列的名字、类型、位置、注释：

•ALTER TABLE table\_name CHANGE [COLUMN] col\_old\_name col\_new\_name column\_type [COMMENT col\_comment] [FIRST|AFTER column\_name]

•这个命令可以允许改变列名、数据类型、注释、列位置或者它们的任意组合

#### 表添加一列 ：

hive> ALTER TABLE pokes ADD COLUMNS (new\_col INT);

#### 添加一列并增加列字段注释

hive> ALTER TABLE invites ADD COLUMNS (new\_col2 INT COMMENT 'a comment');

#### 增加/更新列

•ALTER TABLE table\_name ADD|REPLACE COLUMNS (col\_name data\_type [COMMENT col\_comment], ...)

• ADD是代表新增一字段，字段位置在所有列后面(partition列前)

     REPLACE则是表示替换表中所有字段。

#### 增加表的元数据信息

•ALTER TABLE table\_name SET TBLPROPERTIES table\_properties table\_properties:

         :[property\_name = property\_value…..]

•用户可以用这个命令向表中增加metadata

#### 改变表文件格式与组织

•ALTER TABLE table\_name SET FILEFORMAT file\_format

•ALTER TABLE table\_name CLUSTERED BY(userid) SORTED BY(viewTime) INTO num\_buckets BUCKETS

•这个命令修改了表的物理存储属性

#### 创建／删除视图

•CREATE VIEW [IF NOT EXISTS] view\_name [ (column\_name [COMMENT column\_comment], ...) ][COMMENT view\_comment][TBLPROPERTIES (property\_name = property\_value, ...)] AS SELECT

•增加视图

•如果没有提供表名，视图列的名字将由定义的SELECT表达式自动生成

•如果修改基本表的属性，视图中不会体现，无效查询将会失败

•视图是只读的，不能用LOAD/INSERT/ALTER

•DROP VIEW view\_name

•删除视图

#### 创建数据库

•CREATE DATABASE name

#### 显示命令

•show tables;

•show databases;

•show partitions ;

•show functions

•describe extended table\_name dot col\_name

### DML 操作:元数据存储

     hive不支持用insert语句一条一条的进行插入操作，也不支持update操作。数据是以load的方式加载到建立好的表中。数据一旦导入就不可以修改。

DML包括：INSERT[插入](http://baike.baidu.com/view/226806.htm)、UPDATE[更新](http://baike.baidu.com/view/23174.htm)、DELETE[删除](http://baike.baidu.com/view/14787.htm)

•向数据表内加载文件

•将查询结果插入到Hive表中

•0.8新特性 insert into

#### 向数据表内加载文件

•LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]

•Load 操作只是单纯的复制/移动操作，将数据文件移动到 Hive 表对应的位置。

•filepath

•相对路径，例如：project/data1

•绝对路径，例如： /user/hive/project/data1

•包含模式的完整 URI，例如：hdfs://namenode:9000/user/hive/project/data1

例如：

hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;

#### 加载本地数据，同时给定分区信息

•加载的目标可以是一个表或者分区。如果表包含分区，必须指定每一个分区的分区名

•filepath 可以引用一个文件（这种情况下，Hive 会将文件移动到表所对应的目录中）或者是一个目录（在这种情况下，Hive 会将目录中的所有文件移动至表所对应的目录中）

**LOCAL关键字**

•指定了LOCAL，即本地

•load 命令会去查找本地文件系统中的 filepath。如果发现是相对路径，则路径会被解释为相对于当前用户的当前路径。用户也可以为本地文件指定一个完整的 URI，比如：file:///user/hive/project/data1.

•load 命令会将 filepath 中的文件复制到目标文件系统中。目标文件系统由表的位置属性决定。被复制的数据文件移动到表的数据对应的位置

**例如：加载本地数据，同时给定分区信息：**

hive> LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');

• 没有指定LOCAL

         如果 filepath 指向的是一个完整的 URI，hive 会直接使用这个 URI。 否则

•如果没有指定 schema 或者 authority，Hive 会使用在 hadoop 配置文件中定义的 schema 和 authority，fs.default.name 指定了 Namenode 的 URI

•如果路径不是绝对的，Hive 相对于 /user/ 进行解释。 Hive 会将 filepath 中指定的文件内容移动到 table （或者 partition）所指定的路径中

**加载DFS数据 ，同时给定分区信息：**

hive> LOAD DATA INPATH '/user/myname/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');  
The above command will load data from an HDFS file/directory to the table. Note that loading data from HDFS will result in moving the file/directory. As a result, the operation is almost instantaneous.

#### OVERWRITE

•指定了OVERWRITE

•目标表（或者分区）中的内容（如果有）会被删除，然后再将 filepath 指向的文件/目录中的内容添加到表/分区中。

•如果目标表（分区）已经有一个文件，并且文件名和 filepath 中的文件名冲突，那么现有的文件会被新文件所替代。

#### 将查询结果插入Hive表

•将查询结果插入Hive表

•将查询结果写入HDFS文件系统

•基本模式

     INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select\_statement1 FROM from\_statement

•多插入模式

 FROM from\_statement

INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select\_statement1

[INSERT OVERWRITE TABLE tablename2 [PARTITION ...] select\_statement2] ...

•自动分区模式

 INSERT OVERWRITE TABLE tablename PARTITION (partcol1[=val1], partcol2[=val2] ...) select\_statement FROM from\_statement

#### 将查询结果写入HDFS文件系统

•INSERT OVERWRITE [LOCAL] DIRECTORY directory1 SELECT ... FROM ...

        FROM from\_statement

        INSERT OVERWRITE [LOCAL] DIRECTORY directory1 select\_statement1

     [INSERT OVERWRITE [LOCAL] DIRECTORY directory2 select\_statement2]

•

•数据写入文件系统时进行文本序列化，且每列用^A 来区分，\n换行

#### INSERT INTO

•INSERT INTO  TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select\_statement1 FROM from\_statement

### DQL 操作:数据查询SQL

**SQL操作**

•基本的Select 操作

•基于Partition的查询

•Join

#### 基本的Select 操作

SELECT [ALL | DISTINCT] select\_expr, select\_expr, ...

FROM table\_reference

[WHERE where\_condition]

[GROUP BY col\_list [HAVING condition]]

[   CLUSTER BY col\_list

  | [DISTRIBUTE BY col\_list] [SORT BY| ORDER BY col\_list]

]

[LIMIT number]

•使用ALL和DISTINCT选项区分对重复记录的处理。默认是ALL，表示查询所有记录。DISTINCT表示去掉重复的记录

•

•Where 条件

•类似我们传统SQL的where 条件

•目前支持 AND,OR ,0.9版本支持between

•IN, NOT IN

•不支持EXIST ,NOT EXIST

**ORDER BY与SORT BY的不同**

•ORDER BY 全局排序，只有一个Reduce任务

•SORT BY 只在本机做排序

**Limit**

•Limit 可以限制查询的记录数

SELECT \* FROM t1 LIMIT 5

•实现Top k 查询

•下面的查询语句查询销售记录最大的 5 个销售代表。

SET mapred.reduce.tasks = 1   
  SELECT \* FROM test SORT BY amount DESC LIMIT 5

•REGEX Column Specification

SELECT 语句可以使用正则表达式做列选择，下面的语句查询除了 ds 和 hr 之外的所有列：

SELECT `(ds|hr)?+.+` FROM test

**例如**

#### 按先件查询

hive> SELECT a.foo FROM invites a WHERE a.ds='<DATE>';

#### 将查询数据输出至目录：

hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs\_out' SELECT a.\* FROM invites a WHERE a.ds='<DATE>';

#### 将查询结果输出至本地目录：

hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local\_out' SELECT a.\* FROM pokes a;

#### 选择所有列到本地目录 ：

hive> INSERT OVERWRITE TABLE events SELECT a.\* FROM profiles a;  
hive> INSERT OVERWRITE TABLE events SELECT a.\* FROM profiles a WHERE a.key < 100;  
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/reg\_3' SELECT a.\* FROM events a;  
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg\_4' select a.invites, a.pokes FROM profiles a;  
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg\_5' SELECT COUNT(1) FROM invites a WHERE a.ds='<DATE>';  
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg\_5' SELECT a.foo, a.bar FROM invites a;  
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/sum' SELECT SUM(a.pc) FROM pc1 a;

#### 将一个表的统计结果插入另一个表中：

hive> FROM invites a INSERT OVERWRITE TABLE events SELECT a.bar, count(1) WHERE a.foo > 0 GROUP BY a.bar;  
hive> INSERT OVERWRITE TABLE events SELECT a.bar, count(1) FROM invites a WHERE a.foo > 0 GROUP BY a.bar;  
JOIN  
hive> FROM pokes t1 JOIN invites t2 ON (t1.bar = t2.bar) INSERT OVERWRITE TABLE events SELECT t1.bar, t1.foo, t2.foo;

#### 将多表数据插入到同一表中：

FROM src  
INSERT OVERWRITE TABLE dest1 SELECT src.\* WHERE src.key < 100  
INSERT OVERWRITE TABLE dest2 SELECT src.key, src.value WHERE src.key >= 100 and src.key < 200  
INSERT OVERWRITE TABLE dest3 PARTITION(ds='2008-04-08', hr='12') SELECT src.key WHERE src.key >= 200 and src.key < 300  
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/dest4.out' SELECT src.value WHERE src.key >= 300;

#### 将文件流直接插入文件：

hive> FROM invites a INSERT OVERWRITE TABLE events SELECT TRANSFORM(a.foo, a.bar) AS (oof, rab) USING '/bin/cat' WHERE a.ds > '2008-08-09';  
This streams the data in the map phase through the script /bin/cat (like hadoop streaming). Similarly - streaming can be used on the reduce side (please see the Hive Tutorial or examples)

#### 3.2 基于Partition的查询

•一般 SELECT 查询会扫描整个表，使用 PARTITIONED BY 子句建表，查询就可以利用分区剪枝（input pruning）的特性

•Hive 当前的实现是，只有分区断言出现在离 FROM 子句最近的那个WHERE 子句中，才会启用分区剪枝

#### 3.3 Join

Syntax

join\_table:   
   table\_reference JOIN table\_factor [join\_condition]   
  | table\_reference {LEFT|RIGHT|FULL} [OUTER] JOIN table\_reference join\_condition   
  | table\_reference LEFT SEMI 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 )\*   
  
equality\_expression:   
    expression = expression

•Hive 只支持等值连接（equality joins）、外连接（outer joins）和（left semi joins）。Hive 不支持所有非等值的连接，因为非等值连接非常难转化到 map/reduce 任务

•LEFT，RIGHT和FULL OUTER关键字用于处理join中空记录的情况

•LEFT SEMI JOIN 是 IN/EXISTS 子查询的一种更高效的实现

•join 时，每次 map/reduce 任务的逻辑是这样的：reducer 会缓存 join 序列中除了最后一个表的所有表的记录，再通过最后一个表将结果序列化到文件系统

•实践中，应该把最大的那个表写在最后

**join 查询时，需要注意几个关键点**

•只支持等值join

•SELECT a.\* FROM a JOIN b ON (a.id = b.id)

•SELECT a.\* FROM a JOIN b   
    ON (a.id = b.id AND a.department = b.department)

•可以 join 多于 2 个表，例如

  SELECT a.val, b.val, c.val FROM a JOIN b   
    ON (a.key = b.key1) JOIN c ON (c.key = b.key2)

•如果join中多个表的 join key 是同一个，则 join 会被转化为单个 map/reduce 任务

**LEFT，RIGHT和FULL OUTER**

•例子

•SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key=b.key)

•如果你想限制 join 的输出，应该在 WHERE 子句中写过滤条件——或是在 join 子句中写

•

•容易混淆的问题是表分区的情况

• SELECT c.val, d.val FROM c LEFT OUTER JOIN d ON (c.key=d.key)   
  WHERE a.ds='2010-07-07' AND b.ds='2010-07-07‘

•如果 d 表中找不到对应 c 表的记录，d 表的所有列都会列出 NULL，包括 ds 列。也就是说，join 会过滤 d 表中不能找到匹配 c 表 join key 的所有记录。这样的话，LEFT OUTER 就使得查询结果与 WHERE 子句无关

•解决办法

•SELECT c.val, d.val FROM c LEFT OUTER JOIN d   
  ON (c.key=d.key AND d.ds='2009-07-07' AND c.ds='2009-07-07')

**LEFT SEMI JOIN**

•LEFT SEMI JOIN 的限制是， JOIN 子句中右边的表只能在 ON 子句中设置过滤条件，在 WHERE 子句、SELECT 子句或其他地方过滤都不行

•

•SELECT a.key, a.value   
  FROM a   
  WHERE a.key in   
   (SELECT b.key   
    FROM B);

       可以被重写为：

      SELECT a.key, a.val   
   FROM a LEFT SEMI JOIN b on (a.key = b.key)

**UNION ALL**

•用来合并多个select的查询结果，需要保证select中字段须一致

•select\_statement UNION ALL select\_statement UNION ALL select\_statement ...

### 从SQL到HiveQL应转变的习惯

#### Hive不支持等值连接

•SQL中对两表内联可以写成：

•select \* from dual a,dual b where a.key = b.key;

•Hive中应为

•select \* from dual a join dual b on a.key = b.key;

而不是传统的格式：

SELECT t1.a1 as c1, t2.b1 as c2FROM t1, t2 WHERE t1.a2 = t2.b2

#### 分号字符

•分号是SQL语句结束标记，在HiveQL中也是，但是在HiveQL中，对分号的识别没有那么智慧，例如：

•select concat(key,concat(';',key)) from dual;

•但HiveQL在解析语句时提示：

        FAILED: Parse Error: line 0:-1 mismatched input '<EOF>' expecting ) in function specification

•解决的办法是，使用分号的八进制的ASCII码进行转义，那么上述语句应写成：

•select concat(key,concat('\073',key)) from dual;

#### IS [NOT] NULL

•SQL中null代表空值, 值得警惕的是, 在HiveQL中String类型的字段若是空(empty)字符串, 即长度为0, 那么对它进行IS NULL的判断结果是False.

#### Hive不支持将数据插入现有的表或分区中，

仅支持覆盖重写整个表，示例如下：

**[sql]** [view plaincopyprint?](http://blog.csdn.net/hguisu/article/details/7256833)[在CODE上查看代码片](https://code.csdn.net/snippets/193021)

1. INSERT OVERWRITE TABLE t1
2. SELECT \* FROM t2;

INSERT OVERWRITE TABLE t1

SELECT \* FROM t2;

#### hive不支持INSERT INTO, UPDATE, DELETE操作

    这样的话，就不要很复杂的锁机制来读写数据。  
     INSERT INTO syntax is only available starting in version 0.8。INSERT INTO就是在表或分区中追加数据。

#### hive支持嵌入mapreduce程序，来处理复杂的逻辑

如：

**[sql]** [view plaincopyprint?](http://blog.csdn.net/hguisu/article/details/7256833)[在CODE上查看代码片](https://code.csdn.net/snippets/193021)

1. FROM (
2. MAP doctext USING 'python wc\_mapper.py' AS (word, cnt)
3. FROM docs
4. CLUSTER BY word
5. ) a
6. REDUCE word, cnt USING 'python wc\_reduce.py';

FROM (

MAP doctext USING 'python wc\_mapper.py' AS (word, cnt)

FROM docs

CLUSTER BY word

) a

REDUCE word, cnt USING 'python wc\_reduce.py';

--doctext: 是输入

--word, cnt: 是map程序的输出

--CLUSTER BY: 将wordhash后，又作为reduce程序的输入

并且map程序、reduce程序可以单独使用，如：

**[sql]** [view plaincopyprint?](http://blog.csdn.net/hguisu/article/details/7256833)[在CODE上查看代码片](https://code.csdn.net/snippets/193021)

1. FROM (
2. FROM session\_table
3. SELECT sessionid, tstamp, data
4. DISTRIBUTE BY sessionid SORT BY tstamp
5. ) a
6. REDUCE sessionid, tstamp, data USING 'session\_reducer.sh';

FROM (

FROM session\_table

SELECT sessionid, tstamp, data

DISTRIBUTE BY sessionid SORT BY tstamp

) a

REDUCE sessionid, tstamp, data USING 'session\_reducer.sh';

--DISTRIBUTE BY: 用于给reduce程序分配行数据

#### hive支持将转换后的数据直接写入不同的表，还能写入分区、hdfs和本地目录。

这样能免除多次扫描输入表的开销。

**[sql]** [view plaincopyprint?](http://blog.csdn.net/hguisu/article/details/7256833)[在CODE上查看代码片](https://code.csdn.net/snippets/193021)

1. FROM t1
3. INSERT OVERWRITE TABLE t2
4. SELECT t3.c2, count(1)
5. FROM t3
6. WHERE t3.c1 <= 20
7. GROUP BY t3.c2
9. INSERT OVERWRITE DIRECTORY '/output\_dir'
10. SELECT t3.c2, avg(t3.c1)
11. FROM t3
12. WHERE t3.c1 > 20 AND t3.c1 <= 30
13. GROUP BY t3.c2
15. INSERT OVERWRITE LOCAL DIRECTORY '/home/dir'
16. SELECT t3.c2, sum(t3.c1)
17. FROM t3
18. WHERE t3.c1 > 30
19. GROUP BY t3.c2;

FROM t1

INSERT OVERWRITE TABLE t2

SELECT t3.c2, count(1)

FROM t3

WHERE t3.c1 <= 20

GROUP BY t3.c2

INSERT OVERWRITE DIRECTORY '/output\_dir'

SELECT t3.c2, avg(t3.c1)

FROM t3

WHERE t3.c1 > 20 AND t3.c1 <= 30

GROUP BY t3.c2

INSERT OVERWRITE LOCAL DIRECTORY '/home/dir'

SELECT t3.c2, sum(t3.c1)

FROM t3

WHERE t3.c1 > 30

GROUP BY t3.c2;

### 实际示例

#### 创建一个表

CREATE TABLE u\_data (  
userid INT,  
movieid INT,  
rating INT,  
unixtime STRING)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY '/t'  
STORED AS TEXTFILE;

下载示例数据文件，并解压缩  
wget <http://www.grouplens.org/system/files/ml-data.tar__0.gz>  
tar xvzf ml-data.tar\_\_0.gz

#### 加载数据到表中:

LOAD DATA LOCAL INPATH 'ml-data/u.data'  
OVERWRITE INTO TABLE u\_data;

#### 统计数据总量:

SELECT COUNT(1) FROM u\_data;

#### 现在做一些复杂的数据分析:

创建一个 weekday\_mapper.py: 文件，作为数据按周进行分割   
import sys  
import datetime

for line in sys.stdin:  
line = line.strip()  
userid, movieid, rating, unixtime = line.split('/t')

#### 生成数据的周信息

weekday = datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()  
print '/t'.join([userid, movieid, rating, str(weekday)])

#### 使用映射脚本

//创建表，按分割符分割行中的字段值  
CREATE TABLE u\_data\_new (  
userid INT,  
movieid INT,  
rating INT,  
weekday INT)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY '/t';  
//将python文件加载到系统  
add FILE weekday\_mapper.py;

#### 将数据按周进行分割

INSERT OVERWRITE TABLE u\_data\_new  
SELECT  
TRANSFORM (userid, movieid, rating, unixtime)  
USING 'python weekday\_mapper.py'  
AS (userid, movieid, rating, weekday)  
FROM u\_data;

SELECT weekday, COUNT(1)  
FROM u\_data\_new  
GROUP BY weekday;

#### 处理Apache Weblog 数据

将WEB日志先用正则表达式进行组合，再按需要的条件进行组合输入到表中  
add jar ../build/contrib/hive\_contrib.jar;

CREATE TABLE apachelog (  
host STRING,  
identity STRING,  
user STRING,  
time STRING,  
request STRING,  
status STRING,  
size STRING,  
referer STRING,  
agent STRING)  
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'  
WITH SERDEPROPERTIES (  
"input.regex" = "([^ ]\*) ([^ ]\*) ([^ ]\*) (-|//[[^//]]\*//]) ([^ /"]\*|/"[^/"]\*/") (-|[0-9]\*) (-|[0-9]\*)(?: ([^ /"]\*|/"[^/"]\*/") ([^ /"]\*|/"[^/"]\*/"))?",  
"output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s"  
)  
STORED AS TEXTFILE;

# 附录二：Hive函数

## 关系运算

### 等值比较: =

语法：A=B

操作类型：所有基本类型

 描述: 如果表达式A与表达式B相等，则为TRUE；否则为FALSE

 举例：

 hive>select 1 from lxw\_dual where 1=1;

### 不等值比较: <>

语法: A <> B

操作类型: 所有基本类型

描述: 如果表达式A为NULL，或者表达式B为NULL，返回NULL；如果表达式A与表达式B不相等，则为TRUE；否则为FALSE

举例：

hive> select1 from lxw\_dual where 1 <> 2;

### 小于比较: <

 语法: A < B

操作类型: 所有基本类型

描述: 如果表达式A为NULL，或者表达式B为NULL，返回NULL；如果表达式A小于表达式B，则为TRUE；否则为FALSE

举例：

hive> select1 from lxw\_dual where 1 < 2;

### 小于等于比较: <=

语法: A <= B

操作类型: 所有基本类型

描述: 如果表达式A为NULL，或者表达式B为NULL，返回NULL；如果表达式A小于或者等于表达式B，则为TRUE；否则为FALSE

举例：

hive> select1 from lxw\_dual where 1 <= 1;

1

### 大于比较: >

语法: A > B

操作类型: 所有基本类型

描述: 如果表达式A为NULL，或者表达式B为NULL，返回NULL；如果表达式A大于表达式B，则为TRUE；否则为FALSE

举例：

hive> select1 from lxw\_dual where 2 > 1;

1

### 大于等于比较: >=

语法: A >= B

操作类型: 所有基本类型

描述: 如果表达式A为NULL，或者表达式B为NULL，返回NULL；如果表达式A大于或者等于表达式B，则为TRUE；否则为FALSE

举例：

hive> select1 from lxw\_dual where 1 >= 1;

1

注意：String的比较要注意(常用的时间比较可以先to\_date之后再比较)

hive> select\* from lxw\_dual;

OK

201111120900:00:00     2011111209

hive> selecta,b,a<b,a>b,a=b from lxw\_dual;

201111120900:00:00     2011111209      false  true    false

### 空值判断: IS NULL

语法: A IS NULL

操作类型: 所有类型

描述: 如果表达式A的值为NULL，则为TRUE；否则为FALSE

举例：

hive> select1 from lxw\_dual where null is null;

1

### 非空判断: IS NOTNULL

语法: A IS NOT NULL

操作类型: 所有类型

描述: 如果表达式A的值为NULL，则为FALSE；否则为TRUE

举例：

hive> select1 from lxw\_dual where 1 is not null;

1

### LIKE比较: LIKE

语法: A LIKE B

操作类型: strings

描述: 如果字符串A或者字符串B为NULL，则返回NULL；如果字符串A符合表达式B   的正则语法，则为TRUE；否则为FALSE。B中字符”\_”表示任意单个字符，而字符”%”表示任意数量的字符。

举例：

hive> select1 from lxw\_dual where 'football' like 'foot%';

1

hive> select1 from lxw\_dual where 'football' like 'foot\_\_\_\_';

1

注意：否定比较时候用NOT ALIKE B

hive> select1 from lxw\_dual where NOT 'football' like 'fff%';

1

### JAVA的LIKE操作: RLIKE

语法: A RLIKE B

操作类型: strings

描述: 如果字符串A或者字符串B为NULL，则返回NULL；如果字符串A符合JAVA正则表达式B的正则语法，则为TRUE；否则为FALSE。

举例：

hive> select1 from lxw\_dual where 'footbar’ rlike '^f.\*r$’;

1

注意：判断一个字符串是否全为数字：

hive>select 1from lxw\_dual where '123456' rlike '^\\d+$';

1

hive> select1 from lxw\_dual where '123456aa' rlike '^\\d+$';

### REGEXP操作: REGEXP

语法: A REGEXP B

操作类型: strings

描述: 功能与RLIKE相同

举例：

hive> select1 from lxw\_dual where 'footbar' REGEXP '^f.\*r$';

1

## 数学运算

### 加法操作: +

语法: A + B

操作类型：所有数值类型

说明：返回A与B相加的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。比如，int + int 一般结果为int类型，而int + double 一般结果为double类型

举例：

hive> select1 + 9 from lxw\_dual;

10

hive> createtable lxw\_dual as select 1 + 1.2 from lxw\_dual;

hive>describe lxw\_dual;

\_c0     double

### 减法操作: -

语法: A – B

操作类型：所有数值类型

说明：返回A与B相减的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。比如，int – int 一般结果为int类型，而int – double 一般结果为double类型

举例：

hive> select10 – 5 from lxw\_dual;

5

hive> createtable lxw\_dual as select 5.6 – 4 from lxw\_dual;

hive>describe lxw\_dual;

\_c0     double

### 乘法操作: \*

语法: A \* B

操作类型：所有数值类型

说明：返回A与B相乘的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。注意，如果A乘以B的结果超过默认结果类型的数值范围，则需要通过cast将结果转换成范围更大的数值类型

举例：

hive> select40 \* 5 from lxw\_dual;

200

### 除法操作: /

语法: A / B

操作类型：所有数值类型

说明：返回A除以B的结果。结果的数值类型为double

举例：

hive> select40 / 5 from lxw\_dual;

8.0

注意：hive中最高精度的数据类型是double,只精确到小数点后16位，在做除法运算的时候要特别注意

hive>select ceil(28.0/6.999999999999999999999) from lxw\_duallimit 1;

结果为4

hive>select ceil(28.0/6.99999999999999) from lxw\_dual limit1;

结果为5

### 取余操作: %

语法: A % B

操作类型：所有数值类型

说明：返回A除以B的余数。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。

举例：

hive> select 41 % 5 from lxw\_dual;

1

hive> select 8.4 % 4 from lxw\_dual;

0.40000000000000036

注意：精度在hive中是个很大的问题，类似这样的操作最好通过round指定精度

hive> select round(8.4 % 4 , 2) from lxw\_dual;

0.4

### 位与操作: &

语法: A & B

操作类型：所有数值类型

说明：返回A和B按位进行与操作的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。

举例：

hive> select 4 & 8 from lxw\_dual;

0

hive> select 6 & 4 from lxw\_dual;

4

### 位或操作: |

语法: A | B

操作类型：所有数值类型

说明：返回A和B按位进行或操作的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。

举例：

hive> select 4 | 8 from lxw\_dual;

12

hive> select 6 | 8 from lxw\_dual;

14

### 位异或操作: ^

语法: A ^ B

操作类型：所有数值类型

说明：返回A和B按位进行异或操作的结果。结果的数值类型等于A的类型和B的类型的最小父类型（详见数据类型的继承关系）。

举例：

hive> select 4 ^ 8 from lxw\_dual;

12

hive> select 6 ^ 4 from lxw\_dual;

2

### 位取反操作: ~

语法: ~A

操作类型：所有数值类型

说明：返回A按位取反操作的结果。结果的数值类型等于A的类型。

举例：

hive> select ~6 from lxw\_dual;

-7

hive> select ~4 from lxw\_dual;

-5

## 逻辑运算

### 逻辑与操作: AND

语法: A AND B

操作类型：boolean

说明：如果A和B均为TRUE，则为TRUE；否则为FALSE。如果A为NULL或B为NULL，则为NULL

举例：

hive> select 1 from lxw\_dual where 1=1 and 2=2;

1

### 逻辑或操作: OR

语法: A OR B

操作类型：boolean

说明：如果A为TRUE，或者B为TRUE，或者A和B均为TRUE，则为TRUE；否则为FALSE

举例：

hive> select 1 from lxw\_dual where 1=2 or 2=2;

1

### 逻辑非操作: NOT

语法: NOT A

操作类型：boolean

说明：如果A为FALSE，或者A为NULL，则为TRUE；否则为FALSE

举例：

hive> select 1 from lxw\_dual where not 1=2;

1

## 数值计算

### 取整函数: round

语法: round(double a)

返回值: BIGINT

说明: 返回double类型的整数值部分（遵循四舍五入）

举例：

hive> select round(3.1415926) from lxw\_dual;

3

hive> select round(3.5) from lxw\_dual;

4

hive> create table lxw\_dual as select round(9542.158) fromlxw\_dual;

hive> describe lxw\_dual;

\_c0     bigint

### 指定精度取整函数: round

语法: round(double a, int d)

返回值: DOUBLE

说明: 返回指定精度d的double类型

举例：

hive> select round(3.1415926,4) from lxw\_dual;

3.1416

### 向下取整函数: floor

语法: floor(double a)

返回值: BIGINT

说明: 返回等于或者小于该double变量的最大的整数

举例：

hive> select floor(3.1415926) from lxw\_dual;

3

hive> select floor(25) from lxw\_dual;

25

### 向上取整函数: ceil

语法: ceil(double a)

返回值: BIGINT

说明: 返回等于或者大于该double变量的最小的整数

举例：

hive> select ceil(3.1415926) from lxw\_dual;

4

hive> select ceil(46) from lxw\_dual;

46

### 向上取整函数: ceiling

语法: ceiling(double a)

返回值: BIGINT

说明: 与ceil功能相同

举例：

hive> select ceiling(3.1415926) from lxw\_dual;

4

hive> select ceiling(46) from lxw\_dual;

46

### 取随机数函数: rand

语法: rand(),rand(int seed)

返回值: double

说明: 返回一个0到1范围内的随机数。如果指定种子seed，则会等到一个稳定的随机数序列

举例：

hive> select rand() from lxw\_dual;

0.5577432776034763

hive> select rand() from lxw\_dual;

0.6638336467363424

hive> select rand(100) from lxw\_dual;

0.7220096548596434

hive> select rand(100) from lxw\_dual;

0.7220096548596434

### 自然指数函数: exp

语法: exp(double a)

返回值: double

说明: 返回自然对数e的a次方

举例：

hive> select exp(2) from lxw\_dual;

7.38905609893065

自然对数函数: ln

语法: ln(double a)

返回值: double

说明: 返回a的自然对数

举例：

hive> select ln(7.38905609893065) from lxw\_dual;

2.0

### 以10为底对数函数: log10

语法: log10(double a)

返回值: double

说明: 返回以10为底的a的对数

举例：

hive> select log10(100) from lxw\_dual;

2.0

### 以2为底对数函数: log2

语法: log2(double a)

返回值: double

说明: 返回以2为底的a的对数

举例：

hive> select log2(8) from lxw\_dual;

3.0

### 对数函数: log

语法: log(double base, double a)

返回值: double

说明: 返回以base为底的a的对数

举例：

hive> select log(4,256) from lxw\_dual;

4.0

### 幂运算函数: pow

语法: pow(double a, double p)

返回值: double

说明: 返回a的p次幂

举例：

hive> select pow(2,4) from lxw\_dual;

16.0

### 幂运算函数: power

语法: power(double a, double p)

返回值: double

说明: 返回a的p次幂,与pow功能相同

举例：

hive> select power(2,4) from lxw\_dual;

16.0

### 开平方函数: sqrt

语法: sqrt(double a)

返回值: double

说明: 返回a的平方根

举例：

hive> select sqrt(16) from lxw\_dual;

4.0

### 二进制函数: bin

语法: bin(BIGINT a)

返回值: string

说明: 返回a的二进制代码表示

举例：

hive> select bin(7) from lxw\_dual;

111

### 十六进制函数: hex

语法: hex(BIGINT a)

返回值: string

说明: 如果变量是int类型，那么返回a的十六进制表示；如果变量是string类型，则返回该字符串的十六进制表示

举例：

hive> select hex(17) from lxw\_dual;

11

hive> select hex(‘abc’) from lxw\_dual;

616263

### 反转十六进制函数: unhex

语法: unhex(string a)

返回值: string

说明: 返回该十六进制字符串所代码的字符串

举例：

hive> select unhex(‘616263’)from lxw\_dual;

abc

hive> select unhex(‘11’)from lxw\_dual;

-

hive> select unhex(616263) from lxw\_dual;

abc

### 进制转换函数: conv

语法: conv(BIGINT num, int from\_base, int to\_base)

返回值: string

说明: 将数值num从from\_base进制转化到to\_base进制

举例：

hive> select conv(17,10,16) from lxw\_dual;

11

hive> select conv(17,10,2) from lxw\_dual;

10001

### 绝对值函数: abs

语法: abs(double a)   abs(int a)

返回值: double        int

说明: 返回数值a的绝对值

举例：

hive> select abs(-3.9) from lxw\_dual;

3.9

hive> select abs(10.9) from lxw\_dual;

10.9

### 正取余函数: pmod

语法: pmod(int a, int b),pmod(double a, double b)

返回值: int double

说明: 返回正的a除以b的余数

举例：

hive> select pmod(9,4) from lxw\_dual;

1

hive> select pmod(-9,4) from lxw\_dual;

3

### 正弦函数: sin

语法: sin(double a)

返回值: double

说明: 返回a的正弦值

举例：

hive> select sin(0.8) from lxw\_dual;

0.7173560908995228

### 反正弦函数: asin

语法: asin(double a)

返回值: double

说明: 返回a的反正弦值

举例：

hive> select asin(0.7173560908995228) from lxw\_dual;

0.8

### 余弦函数: cos

语法: cos(double a)

返回值: double

说明: 返回a的余弦值

举例：

hive> select cos(0.9) from lxw\_dual;

0.6216099682706644

### 反余弦函数: acos

语法: acos(double a)

返回值: double

说明: 返回a的反余弦值

举例：

hive> select acos(0.6216099682706644) from lxw\_dual;

0.9

### positive函数: positive

语法: positive(int a), positive(double a)

返回值: int double

说明: 返回a

举例：

hive> select positive(-10) from lxw\_dual;

-10

hive> select positive(12) from lxw\_dual;

12

### negative函数: negative

语法: negative(int a), negative(double a)

返回值: int double

说明: 返回-a

举例：

hive> select negative(-5) from lxw\_dual;

5

hive> select negative(8) from lxw\_dual;

-8

## 日期函数

### UNIX时间戳转日期函数:from\_unixtime

语法: from\_unixtime(bigint unixtime[, string format])

返回值: string

说明: 转化UNIX时间戳（从1970-01-01 00:00:00 UTC到指定时间的秒数）到当前时区的时间格式

举例：

hive> select from\_unixtime(1323308943,'yyyyMMdd') from lxw\_dual;

20111208

### 获取当前UNIX时间戳函数:unix\_timestamp

语法: unix\_timestamp()

返回值: bigint

说明: 获得当前时区的UNIX时间戳

举例：

hive> select unix\_timestamp() from lxw\_dual;

1323309615

### 日期转UNIX时间戳函数:unix\_timestamp

语法: unix\_timestamp(string date)

返回值: bigint

说明: 转换格式为"yyyy-MM-ddHH:mm:ss"的日期到UNIX时间戳。如果转化失败，则返回0。

举例：

hive> select unix\_timestamp('2011-12-07 13:01:03') from lxw\_dual;

1323234063

### 指定格式日期转UNIX时间戳函数:unix\_timestamp

语法: unix\_timestamp(string date, string pattern)

返回值: bigint

说明: 转换pattern格式的日期到UNIX时间戳。如果转化失败，则返回0。

举例：

hive> select unix\_timestamp('20111207 13:01:03','yyyyMMddHH:mm:ss') from lxw\_dual;

1323234063

### 日期时间转日期函数:to\_date

语法: to\_date(string timestamp)

返回值: string

说明: 返回日期时间字段中的日期部分。

举例：

hive> select to\_date('2011-12-08 10:03:01') from lxw\_dual;

2011-12-08

### 日期转年函数: year

语法: year(string date)

返回值: int

说明: 返回日期中的年。

举例：

hive> select year('2011-12-08 10:03:01') from lxw\_dual;

2011

hive> select year('2012-12-08')from lxw\_dual;

2012

### 日期转月函数: month

语法: month (string date)

返回值: int

说明: 返回日期中的月份。

举例：

hive> select month('2011-12-08 10:03:01') from lxw\_dual;

12

hive> select month('2011-08-08')from lxw\_dual;

8

### 日期转天函数: day

语法: day (string date)

返回值: int

说明: 返回日期中的天。

举例：

hive> select day('2011-12-08 10:03:01') from lxw\_dual;

8

hive> select day('2011-12-24')from lxw\_dual;

24

### 日期转小时函数: hour

语法: hour (string date)

返回值: int

说明: 返回日期中的小时。

举例：

hive> select hour('2011-12-08 10:03:01') from lxw\_dual;

10

### 日期转分钟函数: minute

语法: minute (string date)

返回值: int

说明: 返回日期中的分钟。

举例：

hive> select minute('2011-12-08 10:03:01') from lxw\_dual;

3

### 日期转秒函数: second

语法: second (string date)

返回值: int

说明: 返回日期中的秒。

举例：

hive> select second('2011-12-08 10:03:01') from lxw\_dual;

1

### 日期转周函数:weekofyear

语法: weekofyear (string date)

返回值: int

说明: 返回日期在当前的周数。

举例：

hive> select weekofyear('2011-12-08 10:03:01') from lxw\_dual;

49

### 日期比较函数: datediff

语法: datediff(string enddate, string startdate)

返回值: int

说明: 返回结束日期减去开始日期的天数。

举例：

hive> select datediff('2012-12-08','2012-05-09')from lxw\_dual;

213

### 日期增加函数: date\_add

语法: date\_add(string startdate, int days)

返回值: string

说明: 返回开始日期startdate增加days天后的日期。

举例：

hive> select date\_add('2012-12-08',10)from lxw\_dual;

2012-12-18

### 日期减少函数: date\_sub

语法: date\_sub (string startdate, int days)

返回值: string

说明: 返回开始日期startdate减少days天后的日期。

举例：

hive> select date\_sub('2012-12-08',10)from lxw\_dual;

2012-11-28

## 条件函数

### If函数: if

语法: if(boolean testCondition, T valueTrue, T valueFalseOrNull)

返回值: T

说明:  当条件testCondition为TRUE时，返回valueTrue；否则返回valueFalseOrNull

举例：

hive> select if(1=2,100,200) from lxw\_dual;

200

hive> select if(1=1,100,200) from lxw\_dual;

100

### 非空查找函数: COALESCE

语法: COALESCE(T v1, T v2, …)

返回值: T

说明:  返回参数中的第一个非空值；如果所有值都为NULL，那么返回NULL

举例：

hive> select COALESCE(null,'100','50′) from lxw\_dual;

100

### 条件判断函数：CASE

语法: CASE a WHEN b THEN c [WHEN d THEN e]\* [ELSE f] END

返回值: T

说明：如果a等于b，那么返回c；如果a等于d，那么返回e；否则返回f

举例：

hive> Select case 100 when 50 then 'tom' when 100 then 'mary'else 'tim' end from lxw\_dual;

mary

hive> Select case 200 when 50 then 'tom' when 100 then 'mary'else 'tim' end from lxw\_dual;

tim

### 条件判断函数：CASE

语法: CASE WHEN a THEN b [WHEN c THEN d]\* [ELSE e] END

返回值: T

说明：如果a为TRUE,则返回b；如果c为TRUE，则返回d；否则返回e

举例：

hive> select case when 1=2 then 'tom' when 2=2 then 'mary' else'tim' end from lxw\_dual;

mary

hive> select case when 1=1 then 'tom' when 2=2 then 'mary' else'tim' end from lxw\_dual;

tom

## 字符串函数

### 字符串长度函数：length

语法: length(string A)

返回值: int

说明：返回字符串A的长度

举例：

hive> select length('abcedfg') from lxw\_dual;

7

### 字符串反转函数：reverse

语法: reverse(string A)

返回值: string

说明：返回字符串A的反转结果

举例：

hive> select reverse(abcedfg’) from lxw\_dual;

gfdecba

### 字符串连接函数：concat

语法: concat(string A, string B…)

返回值: string

说明：返回输入字符串连接后的结果，支持任意个输入字符串

举例：

hive> select concat(‘abc’,'def’,'gh’) from lxw\_dual;

abcdefgh

### 带分隔符字符串连接函数：concat\_ws

语法: concat\_ws(string SEP, string A, string B…)

返回值: string

说明：返回输入字符串连接后的结果，SEP表示各个字符串间的分隔符

举例：

hive> select concat\_ws(',','abc','def','gh') from lxw\_dual;

abc,def,gh

### 字符串截取函数：substr,substring

语法: substr(string A, int start),substring(string A, int start)

返回值: string

说明：返回字符串A从start位置到结尾的字符串

举例：

hive> select substr('abcde',3) from lxw\_dual;

cde

hive> select substring('abcde',3) from lxw\_dual;

cde

hive>  selectsubstr('abcde',-1) from lxw\_dual;  （和ORACLE相同）

e

### 字符串截取函数：substr,substring

语法: substr(string A, int start, int len),substring(string A, intstart, int len)

返回值: string

说明：返回字符串A从start位置开始，长度为len的字符串

举例：

hive> select substr('abcde',3,2) from lxw\_dual;

cd

hive> select substring('abcde',3,2) from lxw\_dual;

cd

hive>select substring('abcde',-2,2) from lxw\_dual;

de

### 字符串转大写函数：upper,ucase

语法: upper(string A) ucase(string A)

返回值: string

说明：返回字符串A的大写格式

举例：

hive> select upper('abSEd') from lxw\_dual;

ABSED

hive> select ucase('abSEd') from lxw\_dual;

ABSED

### 字符串转小写函数：lower,lcase

语法: lower(string A) lcase(string A)

返回值: string

说明：返回字符串A的小写格式

举例：

hive> select lower('abSEd') from lxw\_dual;

absed

hive> select lcase('abSEd') from lxw\_dual;

absed

### 去空格函数：trim

语法: trim(string A)

返回值: string

说明：去除字符串两边的空格

举例：

hive> select trim(' abc ') from lxw\_dual;

abc

### 左边去空格函数：ltrim

语法: ltrim(string A)

返回值: string

说明：去除字符串左边的空格

举例：

hive> select ltrim(' abc ') from lxw\_dual;

abc

### 右边去空格函数：rtrim

语法: rtrim(string A)

返回值: string

说明：去除字符串右边的空格

举例：

hive> select rtrim(' abc ') from lxw\_dual;

abc

### 正则表达式替换函数：regexp\_replace

语法: regexp\_replace(string A, string B, string C)

返回值: string

说明：将字符串A中的符合java正则表达式B的部分替换为C。注意，在有些情况下要使用转义字符,类似oracle中的regexp\_replace函数。

举例：

hive> select regexp\_replace('foobar', 'oo|ar', '') from lxw\_dual;

fb

### 正则表达式解析函数：regexp\_extract

语法: regexp\_extract(string subject, string pattern, int index)

返回值: string

说明：将字符串subject按照pattern正则表达式的规则拆分，返回index指定的字符。

举例：

hive> select regexp\_extract('foothebar', 'foo(.\*?)(bar)', 1) fromlxw\_dual;

the

hive> select regexp\_extract('foothebar', 'foo(.\*?)(bar)', 2) fromlxw\_dual;

bar

hive> select regexp\_extract('foothebar', 'foo(.\*?)(bar)', 0) fromlxw\_dual;

foothebar

注意，在有些情况下要使用转义字符，下面的等号要用双竖线转义，这是java正则表达式的规则。

select data\_field,

     regexp\_extract(data\_field,'.\*?bgStart\\=([^&]+)',1) as aaa,

     regexp\_extract(data\_field,'.\*?contentLoaded\_headStart\\=([^&]+)',1) as bbb,

     regexp\_extract(data\_field,'.\*?AppLoad2Req\\=([^&]+)',1) as ccc

     from pt\_nginx\_loginlog\_st

     where pt = '2012-03-26'limit 2;

### URL解析函数：parse\_url

语法: parse\_url(string urlString, string partToExtract [, stringkeyToExtract])

返回值: string

说明：返回URL中指定的部分。partToExtract的有效值为：HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO.

举例：

hive> selectparse\_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST') fromlxw\_dual;

facebook.com

hive> selectparse\_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY','k1') from lxw\_dual;

v1

### json解析函数：get\_json\_object

语法: get\_json\_object(string json\_string, string path)

返回值: string

说明：解析json的字符串json\_string,返回path指定的内容。如果输入的json字符串无效，那么返回NULL。

举例：

hive> select get\_json\_object('{"store":

>  {"fruit":\[{"weight":8,"type":"apple"},{"weight":9,"type":"pear"}],

>   "bicycle":{"price":19.95,"color":"red"}

>   },

> "email":"amy@only\_for\_json\_udf\_test.net",

>  "owner":"amy"

> }

> ','$.owner') from lxw\_dual;

amy

### 空格字符串函数：space

语法: space(int n)

返回值: string

说明：返回长度为n的字符串

举例：

hive> select space(10) from lxw\_dual;

hive> select length(space(10)) from lxw\_dual;

10

### 重复字符串函数：repeat

语法: repeat(string str, int n)

返回值: string

说明：返回重复n次后的str字符串

举例：

hive> select repeat('abc',5) from lxw\_dual;

abcabcabcabcabc

### 首字符ascii函数：ascii

语法: ascii(string str)

返回值: int

说明：返回字符串str第一个字符的ascii码

举例：

hive> select ascii('abcde') from lxw\_dual;

97

### 左补足函数：lpad

语法: lpad(string str, int len, string pad)

返回值: string

说明：将str进行用pad进行左补足到len位

举例：

hive> select lpad('abc',10,'td') from lxw\_dual;

tdtdtdtabc

注意：与GP，ORACLE不同，pad 不能默认

### 右补足函数：rpad

语法: rpad(string str, int len, string pad)

返回值: string

说明：将str进行用pad进行右补足到len位

举例：

hive> select rpad('abc',10,'td') from lxw\_dual;

abctdtdtdt

### 分割字符串函数: split

语法:  split(string str, stringpat)

返回值:  array

说明: 按照pat字符串分割str，会返回分割后的字符串数组

举例：

hive> select split('abtcdtef','t') from lxw\_dual;

["ab","cd","ef"]

### 集合查找函数:find\_in\_set

语法: find\_in\_set(string str, string strList)

返回值: int

说明: 返回str在strlist第一次出现的位置，strlist是用逗号分割的字符串。如果没有找该str字符，则返回0

举例：

hive> select find\_in\_set('ab','ef,ab,de') from lxw\_dual;

2

hive> select find\_in\_set('at','ef,ab,de') from lxw\_dual;

0

## 集合统计函数

### 个数统计函数: count

语法: count(\*), count(expr), count(DISTINCT expr[, expr\_.])

返回值: int

说明: count(\*)统计检索出的行的个数，包括NULL值的行；count(expr)返回指定字段的非空值的个数；count(DISTINCTexpr[, expr\_.])返回指定字段的不同的非空值的个数

举例：

hive> select count(\*) from lxw\_dual;

20

hive> select count(distinct t) from lxw\_dual;

10

### 总和统计函数: sum

语法: sum(col), sum(DISTINCT col)

返回值: double

说明: sum(col)统计结果集中col的相加的结果；sum(DISTINCT col)统计结果中col不同值相加的结果

举例：

hive> select sum(t) from lxw\_dual;

100

hive> select sum(distinct t) from lxw\_dual;

70

### 平均值统计函数: avg

语法: avg(col), avg(DISTINCT col)

返回值: double

说明: avg(col)统计结果集中col的平均值；avg(DISTINCT col)统计结果中col不同值相加的平均值

举例：

hive> select avg(t) from lxw\_dual;

50

hive> select avg (distinct t) from lxw\_dual;

30

### 最小值统计函数: min

语法: min(col)

返回值: double

说明: 统计结果集中col字段的最小值

举例：

hive> select min(t) from lxw\_dual;

20

### 最大值统计函数: max

语法: maxcol)

返回值: double

说明: 统计结果集中col字段的最大值

举例：

hive> select max(t) from lxw\_dual;

120

### 非空集合总体变量函数:var\_pop

语法: var\_pop(col)

返回值: double

说明: 统计结果集中col非空集合的总体变量（忽略null）

举例：

### 非空集合样本变量函数:var\_samp

语法: var\_samp (col)

返回值: double

说明: 统计结果集中col非空集合的样本变量（忽略null）

举例：

### 总体标准偏离函数:stddev\_pop

语法: stddev\_pop(col)

返回值: double

说明: 该函数计算总体标准偏离，并返回总体变量的平方根，其返回值与VAR\_POP函数的平方根相同

举例：

### 样本标准偏离函数:stddev\_samp

语法: stddev\_samp (col)

返回值: double

说明: 该函数计算样本标准偏离

举例：

### 中位数函数:percentile

语法: percentile(BIGINT col, p)

返回值: double

说明: 求准确的第pth个百分位数，p必须介于0和1之间，但是col字段目前只支持整数，不支持浮点数类型

举例：

### 中位数函数:percentile

语法: percentile(BIGINT col, array(p1 [, p2]…))

返回值: array<double>

说明: 功能和上述类似，之后后面可以输入多个百分位数，返回类型也为array<double>，其中为对应的百分位数。

举例：

select percentile(score,<0.2,0.4>) from lxw\_dual；取0.2，0.4位置的数据

### 近似中位数函数:percentile\_approx

语法: percentile\_approx(DOUBLE col, p [, B])

返回值: double

说明: 求近似的第pth个百分位数，p必须介于0和1之间，返回类型为double，但是col字段支持浮点类型。参数B控制内存消耗的近似精度，B越大，结果的准确度越高。默认为10,000。当col字段中的distinct值的个数小于B时，结果为准确的百分位数

举例：

### 近似中位数函数:percentile\_approx

语法: percentile\_approx(DOUBLE col, array(p1 [, p2]…) [, B])

返回值: array<double>

说明: 功能和上述类似，之后后面可以输入多个百分位数，返回类型也为array<double>，其中为对应的百分位数。

举例：

### 直方图:histogram\_numeric

语法: histogram\_numeric(col, b)

返回值: array<struct {‘x’,‘y’}>

说明: 以b为基准计算col的直方图信息。

举例：

hive> select histogram\_numeric(100,5) from lxw\_dual;

[{"x":100.0,"y":1.0}]

## 复合类型构建操作

### Map类型构建: map

语法: map (key1, value1, key2, value2, …)

说明：根据输入的key和value对构建map类型

举例：

hive> Create table lxw\_test as select map('100','tom','200','mary')as t from lxw\_dual;

hive> describe lxw\_test;

t      map<string,string>

hive> select t from lxw\_test;

{"100":"tom","200":"mary"}

### Struct类型构建: struct

语法: struct(val1, val2, val3, …)

说明：根据输入的参数构建结构体struct类型

举例：

hive> create table lxw\_test as select struct('tom','mary','tim')as t from lxw\_dual;

hive> describe lxw\_test;

t      struct<col1:string,col2:string,col3:string>

hive> select t from lxw\_test;

{"col1":"tom","col2":"mary","col3":"tim"}

### array类型构建: array

语法: array(val1, val2, …)

说明：根据输入的参数构建数组array类型

举例：

hive> create table lxw\_test as selectarray("tom","mary","tim") as t from lxw\_dual;

hive> describe lxw\_test;

t       array<string>

hive> select t from lxw\_test;

["tom","mary","tim"]

## 复杂类型访问操作

### array类型访问: A[n]

语法: A[n]

操作类型: A为array类型，n为int类型

说明：返回数组A中的第n个变量值。数组的起始下标为0。比如，A是个值为['foo', 'bar']的数组类型，那么A[0]将返回'foo',而A[1]将返回'bar'

举例：

hive> create table lxw\_test as selectarray("tom","mary","tim") as t from lxw\_dual;

hive> select t[0],t[1],t[2] from lxw\_test;

tom     mary    tim

### map类型访问: M[key]

语法: M[key]

操作类型: M为map类型，key为map中的key值

说明：返回map类型M中，key值为指定值的value值。比如，M是值为{'f' -> 'foo', 'b'-> 'bar', 'all' -> 'foobar'}的map类型，那么M['all']将会返回'foobar'

举例：

hive> Create table lxw\_test as selectmap('100','tom','200','mary') as t from lxw\_dual;

hive> select t['200'],t['100'] from lxw\_test;

mary    tom

### struct类型访问: S.x

语法: S.x

操作类型: S为struct类型

说明：返回结构体S中的x字段。比如，对于结构体struct foobar {int foo, int bar}，foobar.foo返回结构体中的foo字段

举例：

hive> create table lxw\_test as select struct('tom','mary','tim')as t from lxw\_dual;

hive> describe lxw\_test;

t      struct<col1:string,col2:string,col3:string>

hive> select t.col1,t.col3 from lxw\_test;

tom     tim

## 复杂类型长度统计函数

### Map类型长度函数: size(Map<K.V>)

语法: size(Map<K.V>)

返回值: int

说明: 返回map类型的长度

举例：

hive> select size(map('100','tom','101','mary')) from lxw\_dual;

2

### array类型长度函数: size(Array<T>)

语法: size(Array<T>)

返回值: int

说明: 返回array类型的长度

举例：

hive> select size(array('100','101','102','103')) from lxw\_dual;

4

### 类型转换函数

类型转换函数: cast

语法: cast(expr as <type>)

返回值: Expected "=" to follow "type"

说明: 返回array类型的长度

举例：

hive> select cast(1 as bigint) from lxw\_dual;

# 附录三：常见问题解决方法

## 问题1

### 错误信息

Error: java.lang.RuntimeException: org.apache.hadoop.hive.ql.metadata.HiveException: java.io.FileNotFoundException: /opt/beh/data/yarn/local/usercache/hadoop/appcache/application\_1419991222083\_0145/container\_1419991222083\_0145\_01\_000007/Stage-6.tar.gz/MapJoin-mapfile62--.hashtable (No such file or directory) at org.apache.hadoop.hive.ql.exec.mr.ExecMapper.map(ExecMapper.java:175) at org.apache.hadoop.mapred.MapRunner.run(MapRunner.java:54) at org.apache.hadoop.mapred.MapTask.runOldMapper(MapTask.java:430) at org.apache.hadoop.mapred.MapTask.run(MapTask.java:342) at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:168) at java.security.AccessController.doPrivileged(Native Method) at javax.security.auth.Subject.doAs(Subject.java:415) at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1548) at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:163)

### 解决办法：

set hive.auto.convert.join=false

## 问题2

### 错误信息

压缩表数据不希望压缩

### 解决办法

set hive.exec.compress.output=false;

## 问题3

### 错误信息

脚本执行内存溢出

### 解决办法

set mapreduce.reduce.memory.mb=7168;

## 问题4

### 错误信息

### 解决办法

## 问题5

### 错误信息

### 解决办法

空值处理：为了保证导出数据的正确性，空值统一用SERDEPROPERTIES('serialization.null.format' = '')进行处理。