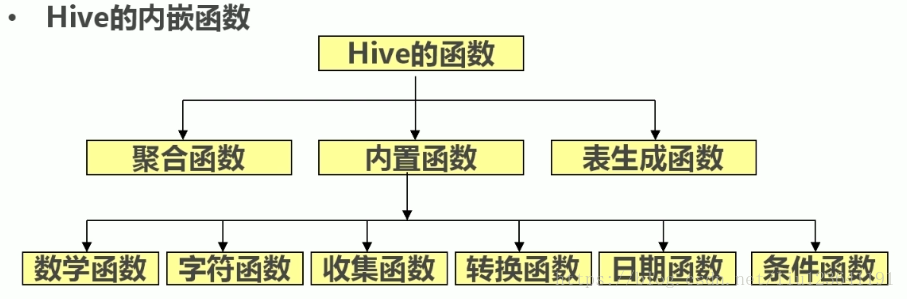
# 1、Hive函数介绍以及内置函数查看



内容较多，见《Hive官方文档》

<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF>

1）查看系统自带的函数

hive> show functions;

2）显示自带的函数的用法

hive> desc function upper;

3）详细显示自带的函数的用法

hive> desc function extended upper;

# 2、常用函数介绍

## 关系运算

### 1、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

|  |
| --- |
| hive> select 1 from tableName where 1=1; |

### 2、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

|  |
| --- |
| hive> select 1 from tableName where 1 <> 2; |

### 3、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

|  |
| --- |
| hive> select 1 from tableName where 1 < 2; |

### 4、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

|  |
| --- |
| hive> select 1 from tableName where 1 < = 1; |

### 5、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

|  |
| --- |
| hive> select 1 from tableName where 2 > 1; |

### 6、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

|  |
| --- |
| hive> select 1 from tableName 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 tableName;  OK  2011111209 00:00:00     2011111209    hive> select a, b, a<b, a>b, a=b from tableName;  2011111209 00:00:00     2011111209      false   true    false |

### 7、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

|  |
| --- |
| hive> select 1 from tableName where null is null; |

### 8、非空判断: IS NOT NULL

**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

|  |
| --- |
| hive> select 1 from tableName where 1 is not null; |

### 9、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。

|  |
| --- |
| hive> select 1 from tableName where 'football' like 'foot%';  hive> select 1 from tableName where 'football' like 'foot\_\_\_\_';  <strong>注意：否定比较时候用NOT A LIKE B</strong>  hive> select 1 from tableName where NOT 'football' like 'fff%'; |

### 10、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。

|  |
| --- |
| hive> select 1 from tableName where 'footbar' rlike '^f.\*r$';  1  注意：判断一个字符串是否全为数字：  hive>select 1 from tableName where '123456' rlike '^\\d+$';  1  hive> select 1 from tableName where '123456aa' rlike '^\\d+$'; |

### 11、REGEXP操作: REGEXP

**语法**: A REGEXP B  
**操作类型**: strings  
**描述**: 功能与RLIKE相同

|  |
| --- |
| hive> select 1 from tableName where 'footbar' REGEXP '^f.\*r$';  1 |

## 数学运算：

### 1、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

|  |
| --- |
| hive> select 1 + 9 from tableName;  10  hive> create table tableName as select 1 + 1.2 from tableName;  hive> describe tableName;  \_c0     double |

### 2、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

|  |
| --- |
| hive> select 10 – 5 from tableName;  5  hive> create table tableName as select 5.6 – 4 from tableName;  hive> describe tableName;  \_c0     double |

### 3、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

|  |
| --- |
| hive> select 40 \* 5 from tableName;  200 |

### 4、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

|  |
| --- |
| hive> select 40 / 5 from tableName;  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 tableName limit 1;  结果为4  hive>select ceil(28.0/6.99999999999999) from tableName limit 1;  结果为5 |

### 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）。

|  |
| --- |
| hive> select 41 % 5 from tableName;  1  hive> select 8.4 % 4 from tableName;  0.40000000000000036  <strong>注意</strong>：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 tableName;  0.4 |

### 6、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）。

|  |
| --- |
| hive> select 4 & 8 from tableName;  0  hive> select 6 & 4 from tableName;  4 |

### 7、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）。

|  |
| --- |
| hive> select 4 | 8 from tableName;  12  hive> select 6 | 8 from tableName;  14 |

### 8、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）。

|  |
| --- |
| hive> select 4 ^ 8 from tableName;  12  hive> select 6 ^ 4 from tableName;  2 |

### 9．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。

|  |
| --- |
| hive> select ~6 from tableName;  -7  hive> select ~4 from tableName;  -5 |

## 逻辑运算：

### 1、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

|  |
| --- |
| hive> select 1 from tableName where 1=1 and 2=2;  1 |

### 2、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

|  |
| --- |
| hive> select 1 from tableName where 1=2 or 2=2;  1 |

### 3、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

|  |
| --- |
| hive> select 1 from tableName where not 1=2;  1 |

## 数值计算

### 1、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）

|  |
| --- |
| hive> select round(3.1415926) from tableName;  3  hive> select round(3.5) from tableName;  4  hive> create table tableName as select round(9542.158) from tableName;  hive> describe tableName;  \_c0     bigint |

### 2、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

|  |
| --- |
| hive> select round(3.1415926,4) from tableName;  3.1416 |

### 3、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

|  |
| --- |
| hive> select floor(3.1415926) from tableName;  3  hive> select floor(25) from tableName;  25 |

### 4、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

|  |
| --- |
| hive> select ceil(3.1415926) from tableName;  4  hive> select ceil(46) from tableName;  46 |

### 5、向上取整函数: ceiling \*\*\*

**语法**: ceiling(double a)  
**返回值**: BIGINT  
**说明**: 与ceil功能相同

|  |
| --- |
| hive> select ceiling(3.1415926) from tableName;  4  hive> select ceiling(46) from tableName;  46 |

### 6、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

|  |
| --- |
| hive> select rand() from tableName;  0.5577432776034763  hive> select rand() from tableName;  0.6638336467363424  hive> select rand(100) from tableName;  0.7220096548596434  hive> select rand(100) from tableName;  0.7220096548596434 |

### 7、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

|  |
| --- |
| hive> select exp(2) from tableName;  7.38905609893065  <strong>自然对数函数</strong>: ln  <strong>语法</strong>: ln(double a)  <strong>返回值</strong>: double  <strong>说明</strong>: 返回a的自然对数  1  hive> select ln(7.38905609893065) from tableName;  2.0 |

### 8、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

|  |
| --- |
| hive> select log10(100) from tableName;  2.0 |

### 9、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

|  |
| --- |
| hive> select log2(8) from tableName;  3.0 |

### 10、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

|  |
| --- |
| hive> select log(4,256) from tableName;  4.0 |

### 11、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

|  |
| --- |
| hive> select pow(2,4) from tableName;  16.0 |

### 12、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

|  |
| --- |
| hive> select power(2,4) from tableName;  16.0 |

### 13、开平方函数: sqrt

**语法**: sqrt(double a)  
**返回值**: double  
**说明**: 返回a的平方根

|  |
| --- |
| hive> select sqrt(16) from tableName;  4.0 |

### 14、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

|  |
| --- |
| hive> select bin(7) from tableName;  111 |

### 15、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

|  |
| --- |
| hive> select hex(17) from tableName;  11  hive> select hex(‘abc’) from tableName;  616263 |

### 16、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

|  |
| --- |
| hive> select unhex(‘616263’) from tableName;  abc  hive> select unhex(‘11’) from tableName;  -  hive> select unhex(616263) from tableName;  abc |

### 17、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

|  |
| --- |
| hive> select conv(17,10,16) from tableName;  11  hive> select conv(17,10,2) from tableName;  10001 |

### 18、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

|  |
| --- |
| hive> select abs(-3.9) from tableName;  3.9  hive> select abs(10.9) from tableName;  10.9 |

### 19、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

|  |
| --- |
| hive> select pmod(9,4) from tableName;  1  hive> select pmod(-9,4) from tableName;  3 |

### 20、正弦函数: sin

**语法**: sin(double a)  
**返回值**: double  
**说明**: 返回a的正弦值

|  |
| --- |
| hive> select sin(0.8) from tableName;  0.7173560908995228 |

### 21、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

|  |
| --- |
| hive> select asin(0.7173560908995228) from tableName;  0.8 |

### 22、余弦函数: cos

**语法**: cos(double a)  
**返回值**: double  
**说明**: 返回a的余弦值

|  |
| --- |
| hive> select cos(0.9) from tableName;  0.6216099682706644 |

### 23、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

|  |
| --- |
| hive> select acos(0.6216099682706644) from tableName;  0.9 |

### 24、positive函数: positive

**语法**: positive(int a), positive(double a)  
**返回值**: int double  
**说明**: 返回a

|  |
| --- |
| hive> select positive(-10) from tableName;  -10  hive> select positive(12) from tableName;  12 |

### 25、negative函数: negative

**语法**: negative(int a), negative(double a)  
**返回值**: int double  
**说明**: 返回-a

|  |
| --- |
| hive> select negative(-5) from tableName;  5  hive> select negative(8) from tableName;  -8 |

## 日期函数

### 1、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

|  |
| --- |
| hive> select from\_unixtime(1323308943,'yyyyMMdd') from tableName;  20111208 |

### 2、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

|  |
| --- |
| hive> select unix\_timestamp() from tableName;  1323309615 |

### 3、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-dd HH:mm:ss"的日期到UNIX时间戳。如果转化失败，则返回0。

|  |
| --- |
| hive> select unix\_timestamp('2011-12-07 13:01:03') from tableName;  1323234063 |

### 4、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。

|  |
| --- |
| hive> select unix\_timestamp('20111207 13:01:03','yyyyMMdd HH:mm:ss') from tableName;  1323234063 |

### 5、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。

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

### 6、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。

|  |
| --- |
| hive> select year('2011-12-08 10:03:01') from tableName;  2011  hive> select year('2012-12-08') from tableName;  2012 |

### 7、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。

|  |
| --- |
| hive> select month('2011-12-08 10:03:01') from tableName;  12  hive> select month('2011-08-08') from tableName;  8 |

### 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。

|  |
| --- |
| hive> select day('2011-12-08 10:03:01') from tableName;  8  hive> select day('2011-12-24') from tableName;  24 |

### 9、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。

|  |
| --- |
| hive> select hour('2011-12-08 10:03:01') from tableName;  10 |

### 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。

|  |
| --- |
| hive> select minute('2011-12-08 10:03:01') from tableName;  3 |

### 11、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。

|  |
| --- |
| hive> select second('2011-12-08 10:03:01') from tableName;  1 |

### 12、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。

|  |
| --- |
| hive> select weekofyear('2011-12-08 10:03:01') from tableName;  49 |

### 13、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。

|  |
| --- |
| hive> select datediff('2012-12-08','2012-05-09') from tableName;  213 |

### 14、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。

|  |
| --- |
| hive> select date\_add('2012-12-08',10) from tableName;  2012-12-18 |

### 15、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。

|  |
| --- |
| hive> select date\_sub('2012-12-08',10) from tableName;  2012-11-28 |

## 条件函数

### 1、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

|  |
| --- |
| hive> select if(1=2,100,200) from tableName;  200  hive> select if(1=1,100,200) from tableName;  100 |

### 2、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

|  |
| --- |
| hive> select COALESCE(null,'100','50') from tableName;  100 |

### 3、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

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

mary

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

tim

### 4、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

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

mary

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

tom

## 字符串函数

### 1、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

|  |
| --- |
| hive> select length('abcedfg') from tableName;  7 |

### 2、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

|  |
| --- |
| hive> select reverse('abcedfg') from tableName;  gfdecba |

### 3、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

|  |
| --- |
| hive> select concat('abc','def','gh') from tableName;  abcdefgh |

### 4、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

|  |
| --- |
| hive> select concat\_ws(',','abc','def','gh')from tableName;  abc,def,gh |

### 5、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

|  |
| --- |
| hive> select substr('abcde',3) from tableName;  cde  hive> select substring('abcde',3) from tableName;  cde  hive>  select substr('abcde',-1) from tableName;  （和ORACLE相同）  e |

### 6、字符串截取函数：substr,substring \*\*\*\*

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

|  |
| --- |
| hive> select substr('abcde',3,2) from tableName;  cd  hive> select substring('abcde',3,2) from tableName;  cd  hive>select substring('abcde',-2,2) from tableName;  de |

### 7、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

|  |
| --- |
| hive> select upper('abSEd') from tableName;  ABSED  hive> select ucase('abSEd') from tableName;  ABSED |

### 8、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

|  |
| --- |
| hive> select lower('abSEd') from tableName;  absed  hive> select lcase('abSEd') from tableName;  absed |

### 9、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

|  |
| --- |
| hive> select trim(' abc ') from tableName;  abc |

### 10、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

|  |
| --- |
| hive> select ltrim(' abc ') from tableName;  abc |

### 11、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

|  |
| --- |
| hive> select rtrim(' abc ') from tableName;  abc |

### 12、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。

|  |
| --- |
| hive> select regexp\_replace('foobar', 'oo|ar', '') from tableName;  fb |

### 13、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。

|  |
| --- |
| hive> select regexp\_extract('foothebar', 'foo(.\*?)(bar)', 1) from tableName;  the  hive> select regexp\_extract('foothebar', 'foo(.\*?)(bar)', 2) from tableName;  bar  hive> select regexp\_extract('foothebar', 'foo(.\*?)(bar)', 0) from tableName;  foothebar  strong>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; |

### 14、URL解析函数：parse\_url \*\*\*\*

**语法**: parse\_url(string urlString, string partToExtract [, string keyToExtract])  
**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.

|  |
| --- |
| hive> select parse\_url  ('[https://www.tableName.com/path1/p.php?k1=v1&k2=v2](https://www.iteblog.com/path1/p.php?k1=v1&k2=v2)#Ref1', 'HOST')  from tableName;  www.tableName.com  hive> select parse\_url  ('[https://www.tableName.com/path1/p.php?k1=v1&k2=v2](https://www.iteblog.com/path1/p.php?k1=v1&k2=v2)#Ref1', 'QUERY', 'k1')  from tableName;  v1 |

### 15、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。

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 tableName;

### 16、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

|  |
| --- |
| hive> select space(10) from tableName;  hive> select length(space(10)) from tableName;  10 |

### 17、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

|  |
| --- |
| hive> select repeat('abc',5) from tableName;  abcabcabcabcabc |

### 18、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

|  |
| --- |
| hive> select ascii('abcde') from tableName;  97 |

### 19、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

|  |
| --- |
| hive> select lpad('abc',10,'td') from tableName;  tdtdtdtabc  注意：与GP，ORACLE不同，pad 不能默认 |

### 20、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

|  |
| --- |
| hive> select rpad('abc',10,'td') from tableName;  abctdtdtdt |

### 21、分割字符串函数: split \*\*\*\*

**语法**: split(string str, string pat)  
**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

|  |
| --- |
| hive> select split('abtcdtef','t') from tableName;  ["ab","cd","ef"] |

### 22、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

|  |
| --- |
| hive> select find\_in\_set('ab','ef,ab,de') from tableName;  2  hive> select find\_in\_set('at','ef,ab,de') from tableName;  0 |

## 集合统计函数

### 1、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(DISTINCT expr[, expr\_.])返回指定字段的不同的非空值的个数

|  |
| --- |
| hive> select count(\*) from tableName;  20  hive> select count(distinct t) from tableName;  10 |

### 2、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

|  |
| --- |
| hive> select sum(t) from tableName;  100  hive> select sum(distinct t) from tableName;  70 |

### 3、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

|  |
| --- |
| hive> select avg(t) from tableName;  50  hive> select avg (distinct t) from tableName;  30 |

### 4、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

|  |
| --- |
| hive> select min(t) from tableName;  20 |

### 5、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

|  |
| --- |
| hive> select max(t) from tableName;  120 |

### 6、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）

### 7、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）

### 8、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

### 9、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

### 10．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

### 11、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。

|  |
| --- |
| select percentile(score,&lt;0.2,0.4>) from tableName； 取0.2，0.4位置的数据 |

### 12、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

### 13、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。

### 14、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。

|  |
| --- |
| hive> select histogram\_numeric(100,5) from tableName;  [{"x":100.0,"y":1.0}] |

## 复合类型构建操作

### 1、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

|  |
| --- |
| hive> Create table mapTable as select map('100','tom','200','mary') as t from tableName;  hive> describe mapTable;  t       map<string ,string>  hive> select t from tableName;  {"100":"tom","200":"mary"} |

### 2、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

|  |
| --- |
| hive> create table struct\_table as select struct('tom','mary','tim') as t from tableName;  hive> describe struct\_table;  t       struct<col1:string ,col2:string,col3:string>  hive> select t from tableName;  {"col1":"tom","col2":"mary","col3":"tim"} |

### 3、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

|  |
| --- |
| hive> create table arr\_table as select array("tom","mary","tim") as t from tableName;  hive> describe tableName;  t       array<string>  hive> select t from tableName;  ["tom","mary","tim"] |

## 复杂类型访问操作 \*\*\*\*

### 1、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'

|  |
| --- |
| hive> create table arr\_table2 as select array("tom","mary","tim") as t  from tableName;  hive> select t[0],t[1] from arr\_table2;  tom     mary    tim |

### 2、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'

|  |
| --- |
| hive> Create table map\_table2 as select map('100','tom','200','mary') as t from tableName;  hive> select t['200'],t['100'] from map\_table2;  mary    tom |

### 3、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

|  |
| --- |
| hive> create table str\_table2 as select struct('tom','mary','tim') as t from tableName;  hive> describe tableName;  t       struct<col1:string ,col2:string,col3:string>  hive> select t.col1,t.col3 from str\_table2;  tom     tim |

## 复杂类型长度统计函数 \*\*\*\*

### 1.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

|  |
| --- |
| hive> select size(t) from map\_table2;  2 |

### 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

|  |
| --- |
| hive> select size(t) from arr\_table2;  4 |

### 3.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**: 返回转换后的数据类型

|  |
| --- |
| hive> select cast('1' as bigint) from tableName;  1 |

# 3、hive当中的lateral view 与 explode以及reflect和窗口函数 \*\*\*\*

## 1、使用explode函数将hive表中的Map和Array字段数据进行拆分

lateral view用于和split、explode等UDTF一起使用的，能将一行数据拆分成多行数据，在此基础上可以对拆分的数据进行聚合，lateral view首先为原始表的每行调用UDTF，UDTF会把一行拆分成一行或者多行，lateral view在把结果组合，产生一个支持别名表的虚拟表。

其中explode还可以用于将hive一列中复杂的array或者map结构拆分成多行

需求：现在有数据格式如下

zhangsan child1,child2,child3,child4 k1:v1,k2:v2

lisi child5,child6,child7,child8 k3:v3,k4:v4

字段之间使用\t分割，需求将所有的child进行拆开成为一列

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

| mychild |

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

| child1 |

| child2 |

| child3 |

| child4 |

| child5 |

| child6 |

| child7 |

| child8 |

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

将map的key和value也进行拆开，成为如下结果

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

| mymapkey | mymapvalue |

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

| k1 | v1 |

| k2 | v2 |

| k3 | v3 |

| k4 | v4 |

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

### 第一步：创建hive数据库

创建hive数据库

hive (default)> create database hive\_explode;

hive (default)> use hive\_explode;

### 第二步：创建hive表，然后使用explode拆分map和array

hive (hive\_explode)> create table t3(name string,children array<string>,address Map<string,string>)

row format delimited fields terminated by '\t'

collection items terminated by ','

map keys terminated by ':' stored as textFile;

### 第三步：加载数据

node03执行以下命令创建表数据文件

mkdir -p /export/servers/hivedatas/

cd /export/servers/hivedatas/

vim maparray

zhangsan child1,child2,child3,child4 k1:v1,k2:v2

lisi child5,child6,child7,child8 k3:v3,k4:v4

hive表当中加载数据

hive (hive\_explode)> load data local inpath '/export/servers/hivedatas/maparray' into table t3;

### 第四步：使用explode将hive当中数据拆开

将array当中的数据拆分开

hive (hive\_explode)> SELECT explode(children) AS myChild FROM t3;

将map当中的数据拆分开

hive (hive\_explode)> SELECT explode(address) AS (myMapKey, myMapValue) FROM t3;

## 2、使用explode拆分json字符串

需求：现在有一些数据格式如下：

a:shandong,b:beijing,c:hebei|1,2,3,4,5,6,7,8,9|[{"source":"7fresh","monthSales":4900,"userCount":1900,"score":"9.9"},{"source":"jd","monthSales":2090,"userCount":78981,"score":"9.8"},{"source":"jdmart","monthSales":6987,"userCount":1600,"score":"9.0"}]

其中字段与字段之间的分隔符是 |

我们要解析得到所有的monthSales对应的值为以下这一列（行转列）

4900

2090

6987

### 第一步：创建hive表

hive (hive\_explode)> create table explode\_lateral\_view

(`area` string,

`goods\_id` string,

`sale\_info` string)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '|'

STORED AS textfile;

### 第二步：准备数据并加载数据

准备数据如下

cd /export/servers/hivedatas

vim explode\_json

a:shandong,b:beijing,c:hebei|1,2,3,4,5,6,7,8,9|[{"source":"7fresh","monthSales":4900,"userCount":1900,"score":"9.9"},{"source":"jd","monthSales":2090,"userCount":78981,"score":"9.8"},{"source":"jdmart","monthSales":6987,"userCount":1600,"score":"9.0"}]

加载数据到hive表当中去

hive (hive\_explode)> load data local inpath '/export/servers/hivedatas/explode\_json' overwrite into table explode\_lateral\_view;

### 第三步：使用explode拆分Array

hive (hive\_explode)> select explode(split(goods\_id,',')) as goods\_id from explode\_lateral\_view;

### 第四步：使用explode拆解Map

hive (hive\_explode)> select explode(split(area,',')) as area from explode\_lateral\_view;

### 第五步：拆解json字段

hive (hive\_explode)> select explode(split(regexp\_replace(regexp\_replace(sale\_info,'\\[\\{',''),'}]',''),'},\\{')) as sale\_info from explode\_lateral\_view;

然后我们想用get\_json\_object来获取key为monthSales的数据：

hive (hive\_explode)> select get\_json\_object(explode(split(regexp\_replace(regexp\_replace(sale\_info,'\\[\\{',''),'}]',''),'},\\{')),'$.monthSales') as sale\_info from explode\_lateral\_view;

然后出现异常FAILED: SemanticException [Error 10081]: UDTF's are not supported outside the SELECT clause, nor nested in expressions

UDTF explode不能写在别的函数内

如果你这么写，想查两个字段，select explode(split(area,',')) as area,good\_id from explode\_lateral\_view;

会报错FAILED: SemanticException 1:40 Only a single expression in the SELECT clause is supported with UDTF's. Error encountered near token 'good\_id'

使用UDTF的时候，只支持一个字段，这时候就需要LATERAL VIEW出场了

## 3、配合LATERAL VIEW使用

配合lateral view查询多个字段

hive (hive\_explode)> select goods\_id2,sale\_info from explode\_lateral\_view LATERAL VIEW explode(split(goods\_id,','))goods as goods\_id2;

其中LATERAL VIEW explode(split(goods\_id,','))goods相当于一个虚拟表，与原表explode\_lateral\_view笛卡尔积关联。

也可以多重使用

hive (hive\_explode)> select goods\_id2,sale\_info,area2

from explode\_lateral\_view

LATERAL VIEW explode(split(goods\_id,','))goods as goods\_id2

LATERAL VIEW explode(split(area,','))area as area2;也是三个表笛卡尔积的结果

最终，我们可以通过下面的句子，把这个json格式的一行数据，完全转换成二维表的方式展现

hive (hive\_explode)> select get\_json\_object(concat('{',sale\_info\_1,'}'),'$.source') as source,

get\_json\_object(concat('{',sale\_info\_1,'}'),'$.monthSales') as monthSales,

get\_json\_object(concat('{',sale\_info\_1,'}'),'$.userCount') as monthSales,

get\_json\_object(concat('{',sale\_info\_1,'}'),'$.score') as monthSales from explode\_lateral\_view

LATERAL VIEW explode(split(regexp\_replace(regexp\_replace(sale\_info,'\\[\\{',''),'}]',''),'},\\{'))sale\_info as sale\_info\_1;

总结：

Lateral View通常和UDTF一起出现，为了解决UDTF不允许在select字段的问题。   
Multiple Lateral View可以实现类似笛卡尔乘积。   
Outer关键字可以把不输出的UDTF的空结果，输出成NULL，防止丢失数据。

## 4、列转行

### 1．相关函数说明

CONCAT(string A/col, string B/col…)：返回输入字符串连接后的结果，支持任意个输入字符串;

CONCAT\_WS(separator, str1, str2,...)：它是一个特殊形式的 CONCAT()。第一个参数剩余参数间的分隔符。分隔符可以是与剩余参数一样的字符串。如果分隔符是 NULL，返回值也将为 NULL。这个函数会跳过分隔符参数后的任何 NULL 和空字符串。分隔符将被加到被连接的字符串之间;

COLLECT\_SET(col)：函数只接受基本数据类型，它的主要作用是将某字段的值进行去重汇总，产生array类型字段。

### 2．数据准备

表6-6 数据准备

|  |  |  |
| --- | --- | --- |
| name | constellation | blood\_type |
| 孙悟空 | 白羊座 | A |
| 老王 | 射手座 | A |
| 宋宋 | 白羊座 | B |
| 猪八戒 | 白羊座 | A |
| 凤姐 | 射手座 | A |

### 3．需求

把星座和血型一样的人归类到一起。结果如下：

射手座,A 老王|凤姐

白羊座,A 孙悟空|猪八戒

白羊座,B 宋宋

### 4．创建本地constellation.txt，导入数据

node03服务器执行以下命令创建文件，注意数据使用\t进行分割

cd /export/servers/hivedatas

vim constellation.txt

孙悟空 白羊座 A

老王 射手座 A

宋宋 白羊座 B

猪八戒 白羊座 A

凤姐 射手座 A

### 5．创建hive表并导入数据

创建hive表并加载数据

hive (hive\_explode)> create table person\_info(

name string,

constellation string,

blood\_type string)

row format delimited fields terminated by "\t";

加载数据

hive (hive\_explode)> load data local inpath '/export/servers/hivedatas/constellation.txt' into table person\_info;

### 6．按需求查询数据

hive (hive\_explode)> select

t1.base,

concat\_ws('|', collect\_set(t1.name)) name

from

(select

name,

concat(constellation, "," , blood\_type) base

from

person\_info) t1

group by

t1.base;

## 5、行专列

### 1．函数说明

EXPLODE(col)：将hive一列中复杂的array或者map结构拆分成多行。

LATERAL VIEW

用法：LATERAL VIEW udtf(expression) tableAlias AS columnAlias

解释：用于和split, explode等UDTF一起使用，它能够将一列数据拆成多行数据，在此基础上可以对拆分后的数据进行聚合。

### 2．数据准备

cd /export/servers/hivedatas

vim movie.txt

数据字段之间使用\t进行分割

《疑犯追踪》 悬疑,动作,科幻,剧情

《Lie to me》 悬疑,警匪,动作,心理,剧情

《战狼2》 战争,动作,灾难

### 3．需求

将电影分类中的数组数据展开。结果如下：

《疑犯追踪》 悬疑

《疑犯追踪》 动作

《疑犯追踪》 科幻

《疑犯追踪》 剧情

《Lie to me》 悬疑

《Lie to me》 警匪

《Lie to me》 动作

《Lie to me》 心理

《Lie to me》 剧情

《战狼2》 战争

《战狼2》 动作

《战狼2》 灾难

### 4．创建hive表并导入数据

创建hive表

create table movie\_info(

movie string,

category array<string>)

row format delimited fields terminated by "\t"

collection items terminated by ",";

加载数据

load data local inpath "/export/servers/hivedatas/movie.txt" into table movie\_info;

### 5．按需求查询数据

select

movie,

category\_name

from

movie\_info lateral view explode(category) table\_tmp as category\_name;

## 6、reflect函数

reflect函数可以支持在sql中调用java中的自带函数，秒杀一切udf函数。

### 使用java.lang.Math当中的Max求两列中最大值

创建hive表

create table test\_udf(col1 int,col2 int) row format delimited fields terminated by ',';

准备数据并加载数据

cd /export/servers/hivedatas

vim test\_udf

1,2

4,3

6,4

7,5

5,6

加载数据

hive (hive\_explode)> load data local inpath '/export/servers/hivedatas/test\_udf' overwrite into table test\_udf;

使用java.lang.Math当中的Max求两列当中的最大值

hive (hive\_explode)> select reflect("java.lang.Math","max",col1,col2) from test\_udf;

### 不同记录执行不同的java内置函数

创建hive表

hive (hive\_explode)> create table test\_udf2(class\_name string,method\_name string,col1 int , col2 int) row format delimited fields terminated by ',';

准备数据

cd /export/servers/hivedatas

vim test\_udf2

java.lang.Math,min,1,2

java.lang.Math,max,2,3

加载数据

hive (hive\_explode)> load data local inpath '/export/servers/hivedatas/test\_udf2' overwrite into table test\_udf2;

执行查询

hive (hive\_explode)> select reflect(class\_name,method\_name,col1,col2) from test\_udf2;

### 判断是否为数字

使用apache commons中的函数，commons下的jar已经包含在hadoop的classpath中，所以可以直接使用。

使用方式如下：

select reflect("org.apache.commons.lang.math.NumberUtils","isNumber","123");

## 7、窗口函数与分析函数

hive当中也带有很多的窗口函数以及分析函数，主要用于以下这些场景

（1）用于分区排序   
（2）动态Group By   
（3）Top N   
（4）累计计算   
（5）层次查询

### 1、创建hive表并加载数据

创建表

hive (hive\_explode)> create table order\_detail(

user\_id string,device\_id string,user\_type string,price double,sales int

)row format delimited fields terminated by ',';

加载数据

cd /export/servers/hivedatas

vim order\_detail

zhangsan,1,new,67.1,2

lisi,2,old,43.32,1

wagner,3,new,88.88,3

liliu,4,new,66.0,1

qiuba,5,new,54.32,1

wangshi,6,old,77.77,2

liwei,7,old,88.44,3

wutong,8,new,56.55,6

lilisi,9,new,88.88,5

qishili,10,new,66.66,5

加载数据

hive (hive\_explode)> load data local inpath '/export/servers/hivedatas/order\_detail' into table order\_detail;

### 2、窗口函数

FIRST\_VALUE：取分组内排序后，截止到当前行，第一个值

LAST\_VALUE： 取分组内排序后，截止到当前行，最后一个值

LEAD(col,n,DEFAULT) ：用于统计窗口内往下第n行值。第一个参数为列名，第二个参数为往下第n行（可选，默认为1），第三个参数为默认值（当往下第n行为NULL时候，取默认值，如不指定，则为NULL）

LAG(col,n,DEFAULT) ：与lead相反，用于统计窗口内往上第n行值。第一个参数为列名，第二个参数为往上第n行（可选，默认为1），第三个参数为默认值（当往上第n行为NULL时候，取默认值，如不指定，则为NULL）

### 3、OVER从句

1、使用标准的聚合函数COUNT、SUM、MIN、MAX、AVG

2、使用PARTITION BY语句，使用一个或者多个原始数据类型的列

3、使用PARTITION BY与ORDER BY语句，使用一个或者多个数据类型的分区或者排序列

4、使用窗口规范，窗口规范支持以下格式：

(ROWS | RANGE) BETWEEN (UNBOUNDED | [num]) PRECEDING AND ([num] PRECEDING | CURRENT ROW | (UNBOUNDED | [num]) FOLLOWING)

(ROWS | RANGE) BETWEEN CURRENT ROW AND (CURRENT ROW | (UNBOUNDED | [num]) FOLLOWING)

(ROWS | RANGE) BETWEEN [num] FOLLOWING AND (UNBOUNDED | [num]) FOLLOWING

1

2

3

当ORDER BY后面缺少窗口从句条件，窗口规范默认是 RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

当ORDER BY和窗口从句都缺失, 窗口规范默认是 ROW BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

OVER从句支持以下函数， 但是并不支持和窗口一起使用它们。

Ranking函数: Rank, NTile, DenseRank, CumeDist, PercentRank.

Lead 和 Lag 函数.

#### 使用窗口函数进行统计求销量

使用窗口函数sum over统计销量

hive (hive\_explode)> select

user\_id,

user\_type,

sales,

--分组内所有行

sum(sales) over(partition by user\_type) AS sales\_1 ,

sum(sales) over(order by user\_type) AS sales\_2 ,

--默认为从起点到当前行，如果sales相同，累加结果相同

sum(sales) over(partition by user\_type order by sales asc) AS sales\_3,

--从起点到当前行，结果与sales\_3不同。 根据排序先后不同，可能结果累加不同

sum(sales) over(partition by user\_type order by sales asc rows between unbounded preceding and current row) AS sales\_4,

--当前行+往前3行

sum(sales) over(partition by user\_type order by sales asc rows between 3 preceding and current row) AS sales\_5,

--当前行+往前3行+往后1行

sum(sales) over(partition by user\_type order by sales asc rows between 3 preceding and 1 following) AS sales\_6,

--当前行+往后所有行

sum(sales) over(partition by user\_type order by sales asc rows between current row and unbounded following) AS sales\_7

from

order\_detail

order by

user\_type,

sales,

user\_id;

统计之后求得结果如下：

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

| user\_id | user\_type | sales | sales\_1 | sales\_2 | sales\_3 | sales\_4 | sales\_5 | sales\_6 | sales\_7 |

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

| liliu | new | 1 | 23 | 23 | 2 | 2 | 2 | 4 | 22 |

| qiuba | new | 1 | 23 | 23 | 2 | 1 | 1 | 2 | 23 |

| zhangsan | new | 2 | 23 | 23 | 4 | 4 | 4 | 7 | 21 |

| wagner | new | 3 | 23 | 23 | 7 | 7 | 7 | 12 | 19 |

| lilisi | new | 5 | 23 | 23 | 17 | 17 | 15 | 21 | 11 |

| qishili | new | 5 | 23 | 23 | 17 | 12 | 11 | 16 | 16 |

| wutong | new | 6 | 23 | 23 | 23 | 23 | 19 | 19 | 6 |

| lisi | old | 1 | 6 | 29 | 1 | 1 | 1 | 3 | 6 |

| wangshi | old | 2 | 6 | 29 | 3 | 3 | 3 | 6 | 5 |

| liwei | old | 3 | 6 | 29 | 6 | 6 | 6 | 6 | 3 |

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

注意:

结果和ORDER BY相关,默认为升序

如果不指定ROWS BETWEEN,默认为从起点到当前行;

如果不指定ORDER BY，则将分组内所有值累加;

关键是理解ROWS BETWEEN含义,也叫做WINDOW子句：

PRECEDING：往前

FOLLOWING：往后

CURRENT ROW：当前行

UNBOUNDED：无界限（起点或终点）

UNBOUNDED PRECEDING：表示从前面的起点

UNBOUNDED FOLLOWING：表示到后面的终点

其他COUNT、AVG，MIN，MAX，和SUM用法一样。

#### 求分组后的第一个和最后一个值first\_value与last\_value

使用first\_value和last\_value求分组后的第一个和最后一个值

select

user\_id,

user\_type,

ROW\_NUMBER() OVER(PARTITION BY user\_type ORDER BY sales) AS row\_num,

first\_value(user\_id) over (partition by user\_type order by sales desc) as max\_sales\_user,

first\_value(user\_id) over (partition by user\_type order by sales asc) as min\_sales\_user,

last\_value(user\_id) over (partition by user\_type order by sales desc) as curr\_last\_min\_user,

last\_value(user\_id) over (partition by user\_type order by sales asc) as curr\_last\_max\_user

from

order\_detail;

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

| user\_id | user\_type | row\_num | max\_sales\_user | min\_sales\_user | curr\_last\_min\_user | curr\_last\_max\_user |

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

| wutong | new | 7 | wutong | qiuba | wutong | wutong |

| lilisi | new | 6 | wutong | qiuba | qishili | lilisi |

| qishili | new | 5 | wutong | qiuba | qishili | lilisi |

| wagner | new | 4 | wutong | qiuba | wagner | wagner |

| zhangsan | new | 3 | wutong | qiuba | zhangsan | zhangsan |

| liliu | new | 2 | wutong | qiuba | qiuba | liliu |

| qiuba | new | 1 | wutong | qiuba | qiuba | liliu |

| liwei | old | 3 | liwei | lisi | liwei | liwei |

| wangshi | old | 2 | liwei | lisi | wangshi | wangshi |

| lisi | old | 1 | liwei | lisi | lisi | lisi |

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

### 4、分析函数

1. ROW\_NUMBER()：

从1开始，按照顺序，生成分组内记录的序列,比如，按照pv降序排列，生成分组内每天的pv名次,ROW\_NUMBER()的应用场景非常多，再比如，获取分组内排序第一的记录;获取一个session中的第一条refer等。

1. RANK() ：

生成数据项在分组中的排名，排名相等会在名次中留下空位

1. DENSE\_RANK() ：

生成数据项在分组中的排名，排名相等会在名次中不会留下空位

1. CUME\_DIST ：

小于等于当前值的行数/分组内总行数。比如，统计小于等于当前薪水的人数，所占总人数的比例

1. PERCENT\_RANK ：

分组内当前行的RANK值-1/分组内总行数-1

1. NTILE(n) ：

用于将分组数据按照顺序切分成n片，返回当前切片值，如果切片不均匀，默认增加第一个切片的分布。NTILE不支持ROWS BETWEEN，比如 NTILE(2) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING AND CURRENT ROW)。

#### RANK、ROW\_NUMBER、DENSE\_RANK OVER的使用

使用这几个函数，可以实现分组求topN

需求：按照用户类型进行分类，求取销售量最大的前N条数据

select

user\_id,user\_type,sales,

RANK() over (partition by user\_type order by sales desc) as r,

ROW\_NUMBER() over (partition by user\_type order by sales desc) as rn,

DENSE\_RANK() over (partition by user\_type order by sales desc) as dr

from

order\_detail;

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

| user\_id | user\_type | sales | r | rn | dr |

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

| wutong | new | 6 | 1 | 1 | 1 |

| qishili | new | 5 | 2 | 2 | 2 |

| lilisi | new | 5 | 2 | 3 | 2 |

| wagner | new | 3 | 4 | 4 | 3 |

| zhangsan | new | 2 | 5 | 5 | 4 |

| qiuba | new | 1 | 6 | 6 | 5 |

| liliu | new | 1 | 6 | 7 | 5 |

| liwei | old | 3 | 1 | 1 | 1 |

| wangshi | old | 2 | 2 | 2 | 2 |

| lisi | old | 1 | 3 | 3 | 3 |

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

#### 使用NTILE求取百分比

我们可以使用NTILE来将我们的数据分成多少份，然后求取百分比

使用NTILE将数据进行分片

select

user\_type,sales,

--分组内将数据分成2片

NTILE(2) OVER(PARTITION BY user\_type ORDER BY sales) AS nt2,

--分组内将数据分成3片

NTILE(3) OVER(PARTITION BY user\_type ORDER BY sales) AS nt3,

--分组内将数据分成4片

NTILE(4) OVER(PARTITION BY user\_type ORDER BY sales) AS nt4,

--将所有数据分成4片

NTILE(4) OVER(ORDER BY sales) AS all\_nt4

from

order\_detail

order by

user\_type,

sales;

得到结果如下：

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

| user\_type | sales | nt2 | nt3 | nt4 | all\_nt4 |

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

| new | 1 | 1 | 1 | 1 | 1 |

| new | 1 | 1 | 1 | 1 | 1 |

| new | 2 | 1 | 1 | 2 | 2 |

| new | 3 | 1 | 2 | 2 | 3 |

| new | 5 | 2 | 2 | 3 | 4 |

| new | 5 | 2 | 3 | 3 | 3 |

| new | 6 | 2 | 3 | 4 | 4 |

| old | 1 | 1 | 1 | 1 | 1 |

| old | 2 | 1 | 2 | 2 | 2 |

| old | 3 | 2 | 3 | 3 | 2 |

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

使用NTILE求取sales前20%的用户id

select

user\_id

from

(select user\_id, NTILE(5) OVER(ORDER BY sales desc) AS nt

from order\_detail

)A

where nt=1;

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

| user\_id |

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

| wutong |

| qishili |

### 5、增强的聚合Cube和Grouping和Rollup

这几个分析函数通常用于OLAP中，不能累加，而且需要根据不同维度上钻和下钻的指标统计，比如，分小时、天、月的UV数。

GROUPING SETS

在一个GROUP BY查询中，根据不同的维度组合进行聚合，等价于将不同维度的GROUP BY结果集进行UNION ALL,

其中的GROUPING\_\_ID，表示结果属于哪一个分组集合。

需求：按照user\_type和sales分别进行分组求取数据

0: jdbc:hive2://node03:10000>select

user\_type,

sales,

count(user\_id) as pv,

GROUPING\_\_ID

from

order\_detail

group by

user\_type,sales

GROUPING SETS(user\_type,sales)

ORDER BY

GROUPING\_\_ID;

求取结果如下：

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

| user\_type | sales | pv | grouping\_\_id |

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

| old | NULL | 3 | 1 |

| new | NULL | 7 | 1 |

| NULL | 6 | 1 | 2 |

| NULL | 5 | 2 | 2 |

| NULL | 3 | 2 | 2 |

| NULL | 2 | 2 | 2 |

| NULL | 1 | 3 | 2 |

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

需求：按照user\_type，sales，以及user\_type + salse 分别进行分组求取统计数据

0: jdbc:hive2://node03:10000>select

user\_type,

sales,

count(user\_id) as pv,

GROUPING\_\_ID

from

order\_detail

group by

user\_type,sales

GROUPING SETS(user\_type,sales,(user\_type,sales))

ORDER BY

GROUPING\_\_ID;

求取结果如下：

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

| user\_type | sales | pv | grouping\_\_id |

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

| old | NULL | 3 | 1 |

| new | NULL | 7 | 1 |

| NULL | 1 | 3 | 2 |

| NULL | 6 | 1 | 2 |

| NULL | 5 | 2 | 2 |

| NULL | 3 | 2 | 2 |

| NULL | 2 | 2 | 2 |

| old | 3 | 1 | 3 |

| old | 2 | 1 | 3 |

| old | 1 | 1 | 3 |

| new | 6 | 1 | 3 |

| new | 5 | 2 | 3 |

| new | 3 | 1 | 3 |

| new | 1 | 2 | 3 |

| new | 2 | 1 | 3 |

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

### 6、使用cube 和ROLLUP 根据GROUP BY的维度的所有组合进行聚合。

#### cube进行聚合

需求：不进行任何的分组，按照user\_type进行分组，按照sales进行分组，按照user\_type+sales进行分组求取统计数据

0: jdbc:hive2://node03:10000>select

user\_type,

sales,

count(user\_id) as pv,

GROUPING\_\_ID

from

order\_detail

group by

user\_type,sales

WITH CUBE

ORDER BY

GROUPING\_\_ID;

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

| user\_type | sales | pv | grouping\_\_id |

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

| NULL | NULL | 10 | 0 |

| new | NULL | 7 | 1 |

| old | NULL | 3 | 1 |

| NULL | 6 | 1 | 2 |

| NULL | 5 | 2 | 2 |

| NULL | 3 | 2 | 2 |

| NULL | 2 | 2 | 2 |

| NULL | 1 | 3 | 2 |

| old | 3 | 1 | 3 |

| old | 2 | 1 | 3 |

| old | 1 | 1 | 3 |

| new | 6 | 1 | 3 |

| new | 5 | 2 | 3 |

| new | 3 | 1 | 3 |

| new | 2 | 1 | 3 |

| new | 1 | 2 | 3 |

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

#### ROLLUP进行聚合

rollup是CUBE的子集，以最左侧的维度为主，从该维度进行层级聚合。

select

user\_type,

sales,

count(user\_id) as pv,

GROUPING\_\_ID

from

order\_detail

group by

user\_type,sales

WITH ROLLUP

ORDER BY

GROUPING\_\_ID;

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

| user\_type | sales | pv | grouping\_\_id |

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

| NULL | NULL | 10 | 0 |

| old | NULL | 3 | 1 |

| new | NULL | 7 | 1 |

| old | 3 | 1 | 3 |

| old | 2 | 1 | 3 |

| old | 1 | 1 | 3 |

| new | 6 | 1 | 3 |

| new | 5 | 2 | 3 |

| new | 3 | 1 | 3 |

| new | 2 | 1 | 3 |

| new | 1 | 2 | 3 |

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

# 4、hive的自定义函数

## 1、Hive自定义函数

1）Hive 自带了一些函数，比如：max/min等，但是数量有限，自己可以通过自定义UDF来方便的扩展。

2）当Hive提供的内置函数无法满足你的业务处理需要时，此时就可以考虑使用用户自定义函数（UDF：user-defined function）。

3）根据用户自定义函数类别分为以下三种：

（1）UDF（User-Defined-Function）

一进一出

（2）UDAF（User-Defined Aggregation Function）

聚集函数，多进一出

类似于：count/max/min

（3）UDTF（User-Defined Table-Generating Functions）

一进多出

如lateral view explore()

4）官方文档地址

https://cwiki.apache.org/confluence/display/Hive/HivePlugins

5）编程步骤：

（1）继承org.apache.hadoop.hive.ql.UDF

（2）需要实现evaluate函数；evaluate函数支持重载；

6）注意事项

（1）UDF必须要有返回类型，可以返回null，但是返回类型不能为void；

（2）UDF中常用Text/LongWritable等类型，不推荐使用java类型；

## 2、UDF开发实例

### 简单UDF示例

#### 第一步：创建maven java 工程，并导入jar包

<repositories>  
 <repository>  
 <id>cloudera</id>  
 <url>https://repository.cloudera.com/artifactory/cloudera-repos/</url>  
 </repository>  
</repositories>  
<dependencies>  
 <dependency>  
 <groupId>org.apache.hadoop</groupId>  
 <artifactId>hadoop-common</artifactId>  
 <version>2.6.0-cdh5.14.0</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.hive</groupId>  
 <artifactId>hive-exec</artifactId>  
 <version>1.1.0-cdh5.14.0</version>  
 </dependency>  
</dependencies>  
<build>  
<plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.0</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 <encoding>UTF-8</encoding>  
 </configuration>  
 </plugin>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-shade-plugin</artifactId>  
 <version>2.2</version>  
 <executions>  
 <execution>  
 <phase>package</phase>  
 <goals>  
 <goal>shade</goal>  
 </goals>  
 <configuration>  
 <filters>  
 <filter>  
 <artifact>\*:\*</artifact>  
 <excludes>  
 <exclude>META-INF/\*.SF</exclude>  
 <exclude>META-INF/\*.DSA</exclude>  
 <exclude>META-INF/\*/RSA</exclude>  
 </excludes>  
 </filter>  
 </filters>  
 </configuration>  
 </execution>  
 </executions>  
 </plugin>  
</plugins>  
</build>

#### 第二步：开发java类继承UDF，并重载evaluate 方法

**public class** KkbUDF **extends** UDF {  
 **public** Text evaluate(**final** Text s) {  
 **if** (**null** == s) {  
 **return null**;  
 }  
 *//返回大写字母* **return new** Text(s.toString().toUpperCase());  
  
 }  
}

#### 第三步：将我们的项目打包，并上传到hive的lib目录下

使用maven的package进行打包，将我们打包好的jar包上传到node03服务器的/export/servers/hive-1.1.0-cdh5.14.0/lib 这个路径下

#### 第四步：添加我们的jar包

重命名我们的jar包名称

cd /export/servers/hive-1.1.0-cdh5.14.0/lib

mv original-day\_06\_hive\_udf-1.0-SNAPSHOT.jar udf.jar

hive的客户端添加我们的jar包

0: jdbc:hive2://node03:10000> add jar /export/servers/hive-1.1.0-cdh5.14.0/lib/udf.jar;

#### 第五步：设置函数与我们的自定义函数关联

0: jdbc:hive2://node03:10000> create temporary function tolowercase as 'cn.kkb.udf.KkbUDF';

#### 第六步：使用自定义函数

0: jdbc:hive2://node03:10000>select tolowercase('abc');

hive当中如何创建永久函数

在hive当中添加临时函数，需要我们每次进入hive客户端的时候都需要添加以下，退出hive客户端临时函数就会失效，那么我们也可以创建永久函数来让其不会失效

创建永久函数

1、指定数据库，将我们的函数创建到指定的数据库下面

0: jdbc:hive2://node03:10000>use myhive;

2、使用add jar添加我们的jar包到hive当中来

0: jdbc:hive2://node03:10000>add jar /export/servers/hive-1.1.0-cdh5.14.0/lib/udf.jar;

3、查看我们添加的所有的jar包

0: jdbc:hive2://node03:10000>list jars;

4、创建永久函数，与我们的函数进行关联

0: jdbc:hive2://node03:10000>create function myuppercase as 'cn.kkb.hive.udf.HiveUDF';

5、查看我们的永久函数

0: jdbc:hive2://node03:10000>show functions like 'my\*';

6、使用永久函数

0: jdbc:hive2://node03:10000>select myhive.myuppercase('helloworld');

7、删除永久函数

0: jdbc:hive2://node03:10000>drop function myhive.myuppercase;

8、查看函数

show functions like 'my\*';

### Json数据解析UDF开发（作业）

作业：

有原始json数据如下：

|  |
| --- |
| {"movie":"1193","rate":"5","timeStamp":"978300760","uid":"1"}  {"movie":"661","rate":"3","timeStamp":"978302109","uid":"1"}  {"movie":"914","rate":"3","timeStamp":"978301968","uid":"1"}  {"movie":"3408","rate":"4","timeStamp":"978300275","uid":"1"}  {"movie":"2355","rate":"5","timeStamp":"978824291","uid":"1"}  {"movie":"1197","rate":"3","timeStamp":"978302268","uid":"1"}  {"movie":"1287","rate":"5","timeStamp":"978302039","uid":"1"} |

需要将数据导入到hive数据仓库中

我不管你中间用几个表，最终我要得到一个结果表：

|  |  |  |  |
| --- | --- | --- | --- |
| movie | rate | timestamp | uid |
| 1197 | 3 | 978302268 | 1 |

注：全在hive中完成，可以用自定义函数

第一步：自定义udf函数，将我们json数据给解析出来，解析成四个字段，整成一个\t分割的一行

第二步：注册我们的自定义函数

第三步：创建一个临时表，加载json格式的数据，加载到临时表里面的一个字段里面去

第四步：insert overwrite local directory 将临时表当中的数据通过我们的自定义函数，给查询出来，放到本地路径下面去

第五步：通过load data的方式，将我们得数据加载到新表当中去