|  |
| --- |
| 无线增值产品部 |
| Hive函数及语法说明 |
| 版本：Hive 0.7.0.001 |

|  |
| --- |
| eagooqi  2011-7-19 |

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 修订人 | 描述 |
| V1.0 | 2010-07-20 | eagooqi | 初稿 |
|  | 2012-1-4 | Eagooqi | ||、cube |

目录

[ 函数说明 2](#_Toc298947106)

[ 内置函数 2](#_Toc298947107)

[ 增加oracle函数 14](#_Toc298947108)

[ 增加业务函数 17](#_Toc298947109)

[ 扩展函数开发规范 17](#_Toc298947110)

[ 语法说明 19](#_Toc298947111)

[ 内置语法 19](#_Toc298947112)

[ 增加语法 27](#_Toc298947113)

[ 扩展语法开发规范（语法转换） 27](#_Toc298947114)

[ ORACLE sql 对应的hSQL语法支持 28](#_Toc298947115)

## 函数说明

参考链接：

*https://cwiki.apache.org/confluence/display/Hive/LanguageManual*

CLI下,使用以下命令显示最新系统函数说明

**SHOW FUNCTIONS;**

**DESCRIBE FUNCTION <function\_name>;**

**DESCRIBE FUNCTION EXTENDED <function\_name>;**

### 内置函数

#### 数值函数Mathematical Functions

|  |  |  |
| --- | --- | --- |
| **返回类型** | **函数名称(参数)** | **描述** |
| BIGINT | round(double a) | 返回double类型的四舍五入的 BIGINT值  Returns the rounded BIGINT value of the double |
| DOUBLE | round(double a, int d) | 返回a的d进制的四舍五入值  Returns the double rounded to d decimal places |
| BIGINT | floor(double a) | 返回小于等于a的最大整数值  Returns the maximum BIGINT value that is equal or less than the double |
| BIGINT | ceil(double a), ceiling(double a) | 返回a的最小的大于等于a的整数值  Returns the minimum BIGINT value that is equal or greater than the double |
| double | rand(), rand(int seed) | Returns a random number (that changes from row to row) that is distributed uniformly from 0 to 1. Specifiying the seed will make sure the generated random number sequence is deterministic. |
| double | exp(double a) | 自然指数函数Returns e^a where e is the base of the natural logarithm |
| double | ln(double a) | 自然对数函数Returns the natural logarithm of the argument |
| double | log10(double a) | 以10为底对数函数Returns the base-10 logarithm of the argument |
| double | log2(double a) | 以2为底对数函数Returns the base-2 logarithm of the argument |
| double | log(double base, double a) | 对数函数Return the base "base" logarithm of the argument |
| double | pow(double a, double p) power(double a, double p) | 幂运算函数Return a^p |
| double | sqrt(double a) | 开平方函数Returns the square root of a |
| string | bin(BIGINT a) | 二进制函数Returns the number in binary format (see [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_bin>]) |
| string | hex(BIGINT a) hex(string a) | 十六进制函数If the argument is an int, hex returns the number as a string in hex format. Otherwise if the number is a string, it converts each character into its hex representation and returns the resulting string. (see [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_hex>]) |
| string | unhex(string a) | 反转十六进制函数Inverse of hex. Interprets each pair of characters as a hexidecimal number and converts to the character represented by the number. |
| string | conv(BIGINT num, int from\_base, int to\_base) | 进制转换函数Converts a number from a given base to another (see [<http://dev.mysql.com/doc/refman/5.0/en/mathematical-functions.html#function_conv>]) |
| double | abs(double a) | 绝对值函数Returns the absolute value |
| int double | pmod(int a, int b) pmod(double a, double b) | 正取余函数Returns the positive value of a mod b |
| double | sin(double a) | 正弦函数Returns the sine of a (a is in radians) |
| double | asin(double a) | 反正弦函数Returns the arc sin of x if -1<=a<=1 or null otherwise |
| double | cos(double a) | 余弦函数Returns the cosine of a (a is in radians) |
| double | acos(double a) | 反余弦函数Returns the arc cosine of x if -1<=a<=1 or null otherwise |
| int double | positive(int a) positive(double a) | positive函数Returns a |
| int double | negative(int a) negative(double a) | negative函数Returns -a |

#### 集合函数Collection Functions

|  |  |  |
| --- | --- | --- |
| **返回类型** | **函数名称(参数)** | **描述** |
| int | size(Map<K.V>) | 返回Map类型的元素个数 |
| int | size(Array<T>) | 返回Array类型的元素个数 |

#### 类型转换函数Type Conversion Functions

|  |  |  |
| --- | --- | --- |
| **返回类型** | **函数名称(参数)** | **描述** |
| **"type"/null** | cast(expr as <type>) | 转换 expr 为 <type> 类型 例如. cast('1' as BIGINT) 转换 string '1' 为整数. 如果转换失败返回 null . |

#### 日期函数Date Functions

|  |  |  |
| --- | --- | --- |
| string | from\_unixtime(bigint unixtime[, string format]) | Converts the number of seconds from unix epoch (1970-01-01 00:00:00 UTC) to a string representing the timestamp of that moment in the current system time zone in the format of "1970-01-01 00:00:00" |
| bigint | unix\_timestamp() | Gets current time stamp using the default time zone. |
| bigint | unix\_timestamp(string date) | Converts time string in format yyyy-MM-dd HH:mm:ss to Unix time stamp, return 0 if fail: unix\_timestamp('2009-03-20 11:30:01') = 1237573801 |
| bigint | unix\_timestamp(string date, string pattern) | Convert time string with given pattern (see [<http://java.sun.com/j2se/1.4.2/docs/api/java/text/SimpleDateFormat.html>]) to Unix time stamp, return 0 if fail: unix\_timestamp('2009-03-20', 'yyyy-MM-dd') = 1237532400 |
| string | to\_date(string timestamp) | Returns the date part of a timestamp string: to\_date("1970-01-01 00:00:00") = "1970-01-01" |
| int | year(string date) | Returns the year part of a date or a timestamp string: year("1970-01-01 00:00:00") = 1970, year("1970-01-01") = 1970 |
| int | month(string date) | Returns the month part of a date or a timestamp string: month("1970-11-01 00:00:00") = 11, month("1970-11-01") = 11 |
| int | day(string date) dayofmonth(date) | Return the day part of a date or a timestamp string: day("1970-11-01 00:00:00") = 1, day("1970-11-01") = 1 |
| int | hour(string date) | Returns the hour of the timestamp: hour('2009-07-30 12:58:59') = 12, hour('12:58:59') = 12 |
| int | minute(string date) | Returns the minute of the timestamp |
| int | second(string date) | Returns the second of the timestamp |
| int | weekofyear(string date) | Return the week number of a timestamp string: weekofyear("1970-11-01 00:00:00") = 44, weekofyear("1970-11-01") = 44 |
| int | datediff(string enddate, string startdate) | Return the number of days from startdate to enddate: datediff('2009-03-01', '2009-02-27') = 2 |
| int | date\_add(string startdate, int days) | Add a number of days to startdate: date\_add('2008-12-31', 1) = '2009-01-01' |
| int | date\_sub(string startdate, int days) | Subtract a number of days to startdate: date\_sub('2008-12-31', 1) = '2008-12-30' |

#### 条件判断函数Conditional Functions

|  |  |  |
| --- | --- | --- |
| T | if(boolean testCondition, T valueTrue, T valueFalseOrNull) | Return valueTrue when testCondition is true, returns valueFalseOrNull otherwise |
| T | COALESCE(T v1, T v2, ...) | Return the first v that is not NULL, or NULL if all v's are NULL |
| T | CASE a WHEN b THEN c [WHEN d THEN e]\* [ELSE f] END | When a = b, returns c; when a = d, return e; else return f |
| T | CASE WHEN a THEN b [WHEN c THEN d]\* [ELSE e] END | When a = true, returns b; when c = true, return d; else return e |

#### 字符串函数String Functions

|  |  |  |
| --- | --- | --- |
| **返回类型** | **函数名称(参数)** | **描述** |
| int | length(string A) | 返回字符串A的长度 |
| string | reverse(string A) | 返回A的反序字符串 |
| string | concat(string A, string B...) | 按照参数顺序返回连接的字符串. 例如. concat('foo', 'bar') 返回 'foobar'. 注:该函数接受任意长度参数. |
| string | concat\_ws(string SEP, string A, string B...) | 如同 concat(), 但间隔连接自定义的分隔符 SEP. |
| string | substr(string A, int start) substring(string A, int start) | 返回 A 的子串从start 位置开始直到结尾，例如. substr('foobar', 4) 返回 'bar' (参见 [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_substr>]) |
| string | substr(string A, int start, int len) substring(string A, int start, int len) | 返回 A 的子串从start 位置开始直到 len长度 例如 substr('foobar', 4, 1) 返回 'b' (参见 [<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_substr>]) |
| string | upper(string A) ucase(string A) | Returns the string resulting from converting all characters of A to upper case e.g. upper('fOoBaR') results in 'FOOBAR' |
| string | lower(string A) lcase(string A) | Returns the string resulting from converting all characters of B to lower case e.g. lower('fOoBaR') results in 'foobar' |
| string | trim(string A) | Returns the string resulting from trimming spaces from both ends of A e.g. trim(' foobar ') results in 'foobar' |
| string | ltrim(string A) | Returns the string resulting from trimming spaces from the beginning(left hand side) of A e.g. ltrim(' foobar ') results in 'foobar ' |
| string | rtrim(string A) | Returns the string resulting from trimming spaces from the end(right hand side) of A e.g. rtrim(' foobar ') results in ' foobar' |
| string | regexp\_replace(string A, string B, string C) | Returns the string resulting from replacing all substrings in B that match the Java regular expression syntax(See Java regular expressions syntax) with C e.g. regexp\_replace("foobar", "oo|ar", "") returns 'fb.' Note that some care is necessary in using predefined character classes: using '\s' as the second argument will match the letter s; ' s' is necessary to match whitespace, etc. |
| string | regexp\_extract(string subject, string pattern, int index) | Returns the string extracted using the pattern. e.g. regexp\_extract('foothebar', 'foo(.\*?)(bar)', 2) returns 'bar.' Note that some care is necessary in using predefined character classes: using '\s' as the second argument will match the letter s; ' s' is necessary to match whitespace, etc. The 'index' parameter is the Java regex Matcher group() method index. See docs/api/java/util/regex/Matcher.html for more information on the 'index' or Java regex group() method. |
| string | parse\_url(string urlString, string partToExtract [, string keyToExtract]) | Returns the specified part from the URL. Valid values for partToExtract include HOST, PATH, QUERY, REF, PROTOCOL, AUTHORITY, FILE, and USERINFO. e.g. parse\_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'HOST') returns 'facebook.com'. Also a value of a particular key in QUERY can be extracted by providing the key as the third argument, e.g. parse\_url('http://facebook.com/path1/p.php?k1=v1&k2=v2#Ref1', 'QUERY', 'k1') returns 'v1'. |
| string | get\_json\_object(string json\_string, string path) | Extract json object from a json string based on json path specified, and return json string of the extracted json object. It will return null if the input json string is invalid. **NOTE: The json path can only have the characters [0-9a-z\_], i.e., no upper-case or special characters. Also, the keys \*cannot** start with numbers.\* This is due to restrictions on Hive column names. |
| string | space(int n) | 返回n 个空格字符串 |
| string | repeat(string str, int n) | 返回n 次重复str连接的字符串 |
| int | ascii(string str) | 返回str的首个字符的ascii码对应数字 |
| string | lpad(string str, int len, string pad) | Returns str, left-padded with pad to a length of len |
| string | rpad(string str, int len, string pad) | Returns str, right-padded with pad to a length of len |
| array | split(string str, string pat) | 按照 pat 正则拆分str (pat是正则表达式) |
| int | find\_in\_set(string str, string strList) | Returns the first occurance of str in strList where strList is a comma-delimited string. Returns null if either argument is null. Returns 0 if the first argument contains any commas. e.g. find\_in\_set('ab', 'abc,b,ab,c,def') returns 3 |
| array<array<string>> | sentences(string str, string lang, string locale) | Tokenizes a string of natural language text into words and sentences, where each sentence is broken at the appropriate sentence boundary and returned as an array of words. The 'lang' and 'locale' are optional arguments. e.g. sentences('Hello there! How are you?') returns ( ("Hello", "there"), ("How", "are", "you") ) |
| array<struct<string,double>> | ngrams(array<array<string>>, int N, int K, int pf) | Returns the top-k N-grams from a set of tokenized sentences, such as those returned by the sentences() UDAF. See [Hive-StatisticsAndDataMining] for more information. |
| array<struct<string,double>> | context\_ngrams(array<array<string>>, array<string>, int K, int pf) | Returns the top-k contextual N-grams from a set of tokenized sentences, given a string of "context". See [Hive-StatisticsAndDataMining] for more information. |

#### 其他函数Misc. Functions

**xpath**

The following functions are described in [Hive-LanguageManual-XPathUDF]:

* xpath, xpath\_short, xpath\_int, xpath\_long, xpath\_float, xpath\_double, xpath\_number, xpath\_string

**get\_json\_object**

A limited version of JSONPath is supported:

* $ : Root object
* . : Child operator
* [] : Subscript operator for array
* \* : Wildcard for []

Syntax not supported that's worth noticing:

* : Zero length string as key
* .. : Recursive descent
* @ : Current object/element
* () : Script expression
* ?() : Filter (script) expression.
* [,] : Union operator
* [start:end.step] : array slice operator

Example: src\_json table is a single column (json), single row table:

json

{"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"

}

The fields of the json object can be extracted using these queries:

hive> SELECT get\_json\_object(src\_json.json, '$.owner') FROM src\_json;

amy

hive> SELECT get\_json\_object(src\_json.json, '$.store.fruit[0]') FROM src\_json;

{"weight":8,"type":"apple"}

hive> SELECT get\_json\_object(src\_json.json, '$.non\_exist\_key') FROM src\_json;

NULL

#### 内置聚合函数Built-in Aggregate Functions (UDAF)

|  |  |  |
| --- | --- | --- |
| **返回类型** | **函数名称(参数)** | **描述** |
| bigint | count(\*), count(expr), count(DISTINCT expr[, expr\_.]) | count(\*) – 返回总行数，包括空值NULL; count(expr) - Returns the number of rows for which the supplied expression is non-NULL; count(DISTINCT expr[, expr]) - Returns the number of rows for which the supplied expression(s) are unique and non-NULL. |
| double | sum(col), sum(DISTINCT col) | Returns the sum of the elements in the group or the sum of the distinct values of the column in the group |
| double | avg(col), avg(DISTINCT col) | Returns the average of the elements in the group or the average of the distinct values of the column in the group |
| double | min(col) | Returns the minimum of the column in the group |
| double | max(col) | Returns the maximum value of the column in the group |
| double | var\_pop(col) | Returns the variance of a numeric column in the group |
| double | var\_samp(col) | Returns the unbiased sample variance of a numeric column in the group |
| double | stddev\_pop(col) | Returns the standard deviation of a numeric column in the group |
| double | stddev\_samp(col) | Returns the unbiased sample standard deviation of a numeric column in the group |
| double | covar\_pop(col1, col2) | Returns the population covariance of a pair of numeric columns in the group |
| double | covar\_samp(col1, col2) | Returns the sample covariance of a pair of a numeric columns in the group |
| double | corr(col1, col2) | Returns the Pearson coefficient of correlation of a pair of a numeric columns in the group |
| double | percentile(col, p) | Returns the exact p^th^ percentile of an integer column in the group (does not work with floating point types). p must be between 0 and 1. |
| array<double> | percentile(col, array(p~1,, [, p,,2,,]...)) | Returns the exact percentiles p,,1,,, p,,2,,, ... of an integer column in the group (does not work with floating point types). p,,i~ must be between 0 and 1. |
| double | percentile\_approx(col, p [, B]) | Returns an approximate p^th^ percentile of a numeric column (including floating point types) in the group. The B parameter controls approximation accuracy at the cost of memory. Higher values yield better approximations, and the default is 10,000. When the number of distinct values in col is smaller than B, this gives an exact percentile value. |
| array<double> | percentile\_approx(col, array(p~1,, [, p,,2\_]...) [, B]) | Same as above, but accepts and returns an array of percentile values instead of a single one. |
| array<struct {'x','y'}> | histogram\_numeric(col, b) | Computes a histogram of a numeric column in the group using b non-uniformly spaced bins. The output is an array of size b of double-valued (x,y) coordinates that represent the bin centers and heights |
| array | collect\_set(col) | Returns a set of objects with duplicate elements eliminated |

### 增加oracle函数

|  |  |  |
| --- | --- | --- |
| **返回类型** | **函数名称(参数)** | **描述** |
| T | Nvl(T1,T2) | 如果T1为空，返回T2，否则返回T1 ；  其中T1 和T2 的类型必须一样。  参数类型同返回值的类型一致。（若参数类型不  一致，函数的行为不可预见，可能执行正确，  可能执行失败） |
| String | s1 || s2 | 如同concat(s1,s2) |
| T | nvl2(T1,T2,T3) | T1 为NULL，则返回T2，否则返回T3；  其中T1、T2、T3 的类型必须一样。参  数类型同返回值的类型一致。 |
| T | decode (T cond, T params[]) | 判断cond是否与params的基数位置的值是否相等，是则返回偶数位置值；否则返回params最后位置的值。 |
| string | systimestamp | 返回当前的系统日期、时间（不支持时区）；输  出时间格式为：yyyy-MM-dd hh24:mi:ss:ff3，当  前仅支持24小时制 |
| string | sysdate | 返回当前的系统日期,输  出时间格式为：yyyy-MM-dd |
| int | INSTR((string,str[,start][,appea  r]) | 返回string 中str 出现的位置。start 代表开始搜  索的位置，可选参数，默认为1（1表示第一个  字符位置）；appear 指示搜索第几次出现的，可  选参数，默认为1；若没有匹配到，返回0。（ start  和appear 取值为大于0的整数） |
| int/long/floa  t/double | Mod（n1,n2） | 返回一个n1 除以n2 的余数。支持的类型有  Byte、Short、Integer、Long、Float、Double。  返回值的正负和n1 相关。使用此函数需要注意  的2 个问题：  1、对小数执行mod 计算可能会产生精度丢失，  （如  mod(3.1415926535897384626,3.1415926535897384  627，返回结果为0.0)  ）此问题由java 环境本身导致，若需要高精度  的浮点计算，慎用此函数；  2、传入比MAX\_LONG 还大的整数作为参数，  则参数会被自动升级成Double 类型，函数也可  以正常计算结果，但返回的结果是小数类型。 |
| int | Bitand(expr1,expr2) | 指定按位进行AND 运算的两个数值。当前只  支持Byte、Short、Integer、Long 类型。如果expr1  和expr2的位都是1，相应的结果位就是1;否  则相应的结果位是0。若expr1和expr2类型不  一样，直接进行and 计算 |
| string | next\_day(date,day) | 计算给出日期date 之后的下一个星期day 的日  期。day 是数字， 1-7分别表示星期日-六；返  回日期的格式为“YYYY-MM-DD” |
| int | to\_number(string) | 将给出的字符转换为数字；string 必须为全数字  串。（ Oracle 中的to\_number 很复杂，可变参数  且支持多种类型。当前仅支持整数类  型，包括short、int、long） |
| string | to\_char(date,format) | 将日期date 转化为一个字符串；date 的格式固  定为yyyy-mm-dd hh24:mi:ss:ff3，输出的格式由  format 指定。format 当前支持的格式如下（不  区分大小写）：  yyyymmdd， 年月日；  yyyymm，年月；  mm，月  dd，日  yyyy-mm-dd  yyyy-mm  yyyymmddhh24miss，年月日时分秒（24小时制）  yyyy-mm-dd hh24:mi:ss  hh24miss  yyyymmddhh24missff3，年月日时分秒毫秒（24  小时制） |
| string | to\_date(date,[format]) | 将字符串转化为一个日期；format 指定了date  的解析方式，支持的格式同to\_char。若没有指  定format，缺省的格式为：yyyy-mm-dd |
| double/int/s  tring | least( expr1, expr2, ... expr\_n ) | Returns the smallest value in a list of expressions.  expr1, expr2, . expr\_n are expressions that are  evaluated by the least function.  Having a NULL value in one of the expressions  will return NULL as the least value.  目前支持int，double 和string 三种类型，如果  不同类型出现在同一组参数中会报错。  如果比较的是字符串，暂不支持转义符，如果  字符串中含有"/"需要写成"//"。 |
| string | date\_sub2(string startdate, int days) | 对date\_sub的扩展，startdate 支持yyyyMM、yyyyMMdd、yyyyMMddHH、yyyyMMddHHmm日期时间格式；按照最小单位回减。（支持负数） |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

### 增加业务函数

|  |  |  |
| --- | --- | --- |
| **返回类型** | **函数名称(参数)** | **描述** |
| Map | url\_to\_map(string url [,string p1,string p2 ]） | 根据书城业务的url，返回分解的url参数键/值对为map类型,参数可选，默认p1为&，p2为=  拆分url,如果aid为空，则设置aid=uri  增加key为url\_uri的键 |
| Map | url\_to\_map2(string url [,string p1,string p2 ]） | 与url\_to\_map区别是此函数取第一个参数的值。 |
| Map | Qua(String qua) | 业务函数：取qua字段,返回对应的qua和model字段  没取到返回-  如qua(‘SQB30\_GA/30793&SMTT\_3/020100&SYM5&402214&NokiaC5-03&0&8865&V3’)[‘qua’] = ‘SQB30\_GA’ |
| string | Ua(String qua,String ua) | 根据qua,ua参数返回对应平台如:android,symbian  没匹配到返回unknown  没记录返回NoUa |
| Map | Ua2(string qua,string ua) | 根据qua,ua 参数返回平台、浏览器；  Platform、browser  如:ua2(qua,ua)[‘platform’] =’Android’  ua2(qua,ua)[‘browser’] =’mqqbrowser’ |
| Map | Ipinfo(String ip) | 根据ip地址返回相应信息包括：apn、city、country、operator、province  如:ipinfo(‘221.34.23.44’)[‘province’]=’北京’ |
| String | Cd\_f(cd\_f,entry\_cd\_f,qua) | 业务函数,根据cd\_f,entry\_cd\_f,qua取的渠道信息 |
|  |  |  |
|  |  |  |

## 扩展函数开发规范

提供以下两种实现方式：

##### a继承org.apache.hadoop.hive.ql.exec.UDF类

代码包为:package org.apache.hadoop.hive.ql.udf

实现evaluate方法 ，根据输入参数和返回参数类型，系统自动转换到匹配的方法实现上。

例如：

* UDFTestLength.java:

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

public class UDFTestLength extends UDF {

IntWritable result = new IntWritable();

public IntWritable evaluate(Text s) {

if (s == null) {

return null;

}

result.set(countUDF8Characters(s));

return result;

}

}

##### b继承org.apache.hadoop.hive.ql.udf.generic. GenericUDF类

代码包为：package org.apache.hadoop.hive.ql.udf. generic

实现initialize ，evaluate， getDisplayString方法

例如：

@Description(name = "url\_to\_map", value = "\_FUNC\_(text, delimiter1, delimiter2) - "

**public** **class** GenericUDFUrlToMap **extends** GenericUDF {

HashMap<Object, Object> ret = **new** HashMap<Object, Object>();

@Override

**public** ObjectInspector initialize(ObjectInspector[] arguments)

**throws** UDFArgumentException {

… …

**return** ObjectInspectorFactory.*getStandardMapObjectInspector*(

PrimitiveObjectInspectorFactory.*javaStringObjectInspector*,

PrimitiveObjectInspectorFactory.*javaStringObjectInspector*);

}

@Override

**public** Object evaluate(DeferredObject[] arguments) **throws** HiveException {

ret.clear();

… …

**return** ret;

}

@Override

**public** String getDisplayString(String[] children) {

StringBuilder sb = **new** StringBuilder();

sb.append("url\_to\_map(");

**assert** (children.length <= 3);

**boolean** firstChild = **true**;

**for** (String child : children) {

**if** (firstChild) {

firstChild = **false**;

} **else** {

sb.append(",");

}

sb.append(child);

}

sb.append(")");

**return** sb.toString();

}

}

扩展函数写完后需要统一注册到FunctionRegistry中，发布后成为系统函数；

## 语法说明

### 内置语法

#### Select 语法

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]

* SELECT 语句可以是[union] 查询或 [subquery] 子查询的一部分.
* table\_reference 是查询的输入项.他可以是一个 table, 或者[ view], 或 [join construct] 或者 [subquery].
* 简单查询. 下面的例子检索 table t1的所有列所有行.
* SELECT \* FROM t1

**WHERE 子句**

Where条件是 [boolean] [expression].例如,下列查询只列出 amount 大于 10 并且是 US region的sales. 在 WHERE 子句中，Hive 不支持 IN, EXISTS 或子查询subqueries.

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

**ALL / DISTINCT 子句**

ALL DISTINCT 选项标识是否换回重复的行记录. 如果无标识，默认使用ALL (返回所有匹配到的行). DISTINCT标识返回去重的结果集.

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

**基于分区的查询Partition Based Queries**

一般而言, SELECT 查询检索整个表记录 (不同于[sampling]). 如果建表时使用 [PARTITIONED BY] 子句, 查询时会使用分区裁剪 **partition pruning** 只检索相关分区的数据. 如果在WHERE子句或者 JOIN的 ON 子句中使用分区，Hive 便使用分区裁剪.例如, 表 page\_views 列 date是分区, 以下查询行只在日期2008-03-01 到 2008-03-31之间.

SELECT page\_views.\*

FROM page\_views

WHERE page\_views.date >= '2008-03-01' AND page\_views.date <= '2008-03-31'

如果表 page\_views 被另一个表 dim\_users连接 ,你可以在ON子句中标记一个分区范围.

SELECT page\_views.\*

FROM page\_views JOIN dim\_users

ON (page\_views.user\_id = dim\_users.id AND page\_views.date >= '2008-03-01' AND page\_views.date <= '2008-03-31')

* 也可参见 [Group By]
* 也可参见 [Sort By / Cluster By / Distribute By / Order By]

**HAVING 子句**

Hive 0.7.0 已经支持 HAVING 子句. 也可以用subquery子查询替换实现.例如,

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

可以用以下替换

SELECT col1 FROM (SELECT col1, SUM(col2) AS col2sum FROM t1 GROUP BY col1) t2 WHERE t2.col2sum > 10

**LIMIT 子句**

Limit 返回指定数量的行. 返回的行是随机的. 以下查询是随机返回5行记录.

SELECT \* FROM t1 LIMIT 5

* Top k 查询. 以下查询结果是top 5 sales 记录按 amount 倒序.
* SET mapred.reduce.tasks = 1
* SELECT \* FROM sales SORT BY amount DESC LIMIT 5

**REGEX Column Specification**

A SELECT statement can take regex-based column specification.

* We use java regex syntax. Try <http://www.fileformat.info/tool/regex.htm> for testing purposes.
* The following query select all columns except ds and hr.
* SELECT {{(ds|hr)?+.+}} FROM sales

#### 使用insert子句将查询结果插入到hive表（Inserting data into Hive Tables from queriesQuery Results can be inserted into tables by using the insert clause）

**语法**

标准语法:

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

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

Hive 扩展 (多条插入multiple inserts):

FROM from\_statement

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

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

[INSERT INTO TABLE tablename2 [PARTITION ...] select\_statement2] ...;

FROM from\_statement

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

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

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

Hive extension (dynamic partition inserts):

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

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

#### Join 语法

Hive 支持以下join语法:

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

只有等值连接 equality joins,外连接 outer joins, 和左等值连接 left semi joins 是在 Hive中支持的. 非等值连接不支持，支持多表连接.

考虑以下情况的连接使用:

* 只有等值连接可以使用.
* 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)

都是合法连接,但

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

是不正确的。

* 多个表连接是可以的，如.
* SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2)

是合法连接.

* Hive会转换多个表为一个map/reduce作业，如果连接键是同一个列 ，如.
* SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1)

is converted into a single map/reduce job as only key1 column for b is involved in the join. 另外

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

会被转换为2个 map/reduce 作业，因为第一次使用 key1 作为条件，第二次使用 key2 作为条件. 第一个 map/reduce 作业连接 a 和 b ，结果连接c作为第二个 map/reduce 作业.

* 在每个 map/reduce 连接阶段, 在reducers 里， 按顺序的最后一个表被 streamed 其他表被缓存 buffered. 因此，把大表放到顺序的最后，使他帮助减少内存，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)

所有的3张表被放到一个 map/reduce 作业中 并且在 reducers过程中表a 和 b的key value被缓存到内存中，检索c的每一行记录时，连接直接在缓存中进行计算.如同

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

这里有2个 map/reduce作业解决这个连接. 第一个作业连接 a 和 b ，当在reducers中处理b的行时，先缓存表a. 第二个作业先缓存第一个的结果集，然后在reduces中遍历c时

* 在 map/reduce的 join阶段, 可以通过语法提示hint设定表是否被 streamed. 如
* SELECT /\*+ STREAMTABLE(a) \*/ a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1)

3个表在一个map/reduce作业中执行，先缓存b和c到缓存中，再遍历a表的记录.

* LEFT, RIGHT, and FULL OUTER joins 对未匹配的行提供更多控制机制.例如:
* SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON (a.key=b.key)

返回a表的所有记录， 结果行是 a.val,b.val 当b.key 等于 a.key, 并且结果行是 a.val,NULL 当没有相应的b.key.b中没有相应的a.key的被过滤掉. 语法"FROM a LEFT OUTER JOIN b" 必须写在同一行里，以便理解工作机制--a 是 LEFT b在查询中,故 a表所有的行被保留; a RIGHT OUTER JOIN 将保留b的所有行, FULL OUTER JOIN 将保留所有a和b的行，OUTER JOIN应符合标准的SQL规格语义

* Joins 发生在 WHERE CLAUSES之前. 所以, 应该在join子句里限制条件过滤记录，尤其是在分区表里:
* 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'

连接a和b产生 a.val b.val结果集. WHERE子句可以过滤记录，然而a的行中包括ds列都将成为NULL.也就是说,将过滤不到记录. 但用以下语法:

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')

..连接的输出结果被提前过滤, 不会出现过滤到没匹配b.key的记录行. 同样的逻辑适应 RIGHT 和 FULL joins.

* 连接是不可交换的！连接是左结合的，不管他们是否左连接或右连接
* 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)

...首先在a上连接b, 所有的a或b，在其他表中没有相应的键。Reducer的结果表，连接C.这提供了直观的结果，如果有一个key存在A和C，但不是B：整个行（其中包括a.val1，a.val2，和a.key）在“A JOIN B”，中被过滤掉，因为他不不在b中，结果集中没有a.key,所以当是LEFT OUT JOIN c的时候，c.val不会出现，因为没有c.key匹配到a.key（因为a中的行被过滤）。同样，如果这是右外连接（而不是左连接），我们将得到怪异的效果：为NULL，NULL，NULL，c.val，因为我们指定了作为连接条件的a.key= c.key，过滤掉了不匹配第一个连接的所有行。  
为了达到更直观的效果，我们应该使用FROM c LEFT OUTER JOIN a ON (c.key = a.key) LEFT OUTER JOIN b ON (c.key = b.key)

* LEFT SEMI JOIN更高效得实现了IN/EXISTS subquery语法. 虽然 Hive 目前不支持 IN/EXISTS 子查询，但你可以改写为 LEFT SEMI JOIN. LEFT SEMI JOIN使用限制是只能是表右边的连接(ON-clause),不能在WHERE-或 SELECT-子句中使用.
* SELECT a.key, a.value

FROM a

WHERE a.key in

(SELECT b.key FROM B);

可以写成:

SELECT a.key, a.val

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

* 如果除了被连接的表A是大表外，其它的都是小表, 可以在join优化，只在mapper中执行. 以下查询
* SELECT /\*+ MAPJOIN(b) \*/ a.key, a.value

FROM a join b on a.key = b.key

不需要reducer. 对于A的每个mapper, B 被完全读入内存.限制是 **FULL/RIGHT OUTER JOIN b** 不能被优化。

* 如果连接表是 bucketized, 并且buckets都是多个的, buckets可以互相连接. 如 A 有 8 个buckets 表 B 有 4个 buckets,下面的连接
* SELECT /\*+ MAPJOIN(b) \*/ a.key, a.value

FROM a join b on a.key = b.key

只在mapper中执行. B 不用为每个A的mapper完全读入 ,只需要读取需要的buckets. 对上面的查询, A的bucket 1 mapper 处理时只读取B的bucket 1 . 这不是默认设置,需要设置下列环境变量

**set hive.optimize.bucketmapjoin = true**

* 如果表都被 sorted 和bucketized,并且bucket数量一致, sort-merge join优化被使用. 相关的buckets在mapper中被连接. 如 A 和 B 都是 4个 buckets,
* SELECT /\*+ MAPJOIN(b) \*/ a.key, a.value

FROM A a join B b on a.key = b.key

只在mapper中执行. A中的 bucket对应的mapper检索相关的B中的bucket. 这不是默认设置,需要设置下列环境变量:

**set hive.input.format=org.apache.hadoop.hive.ql.io.BucketizedHiveInputFormat;**

**set hive.optimize.bucketmapjoin = true;**

**set hive.optimize.bucketmapjoin.sortedmerge = true;**

#### Subquery 子查询语法

SELECT ... FROM (subquery) name ...

Hive 只在 FROM 子句里支持子查询. 子查询语句必须给定一个别名. 子查询中的列名必须唯一. 像表的列一样，子查询中的列一样可以被引用. 子查询也可以是 UNION的一部分. Hive 支持多级子查询嵌套.

子查询简单例子:

SELECT col

FROM (

SELECT a+b AS col

FROM t1

) t2

子查询包含在 UNION ALL中:

SELECT t3.col

FROM (

SELECT a+b AS col

FROM t1

UNION ALL

SELECT c+d AS col

FROM t2

) t3

### 增加语法

对with子句的支持，自动转换为subquery子句。

实现hive.semantic.analyzer.hook

org.apache.hadoop.hive.ql.metadata.WithSemanticAnalyzerHook

*后续增加对with语法中间表的优化*

对charset的支持

例如：

ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n' with ("charset"="gbk");

或者

ALTER TABLE table\_name SET SERDEPROPERTIES ('charset'= 'gbk');

对cube、rollup、grouping的支持

实现hive.semantic.analyzer.hook

org.apache.hadoop.hive.ql.metadata.RCSemanticAnalyzerHook

## 扩展语法开发规范（语法转换）

修改hive.g增加关键词法和语义解析。

包存放在：org.apache.hadoop.hive.ql.metadata

具体实现继承org.apache.hadoop.hive.ql.parse.AbstractSemanticAnalyzerHook

实现preAnalyze 和postAnalyze方法

修改hive-site.xml配置，增加hook实现

## ORACLE sql 对应的hSQL语法支持

|  |
| --- |
|  |
|
| ORACLE SQL | | HSQL | |  |
| FROM | | 支持 | from后可以跟表，子查询，union，join等 |  |
| CASE | | 支持 | CASE a WHEN b THEN c [WHEN d THEN e]\* [ELSE f] END 和 CASE WHEN a THEN b [WHEN c THEN d]\* [ELSE e] END |  |
| SELECT | | 支持 | SELECT [ALL | DISTINCT] select\_expr, select\_expr, ...  select\_expr可以是常量，列，函数等 |  |
| WHERE | | 支持 | WHERE where\_condition where\_condition是任何返回true或者false的表达式 |  |
| NULL | | 支持 | select null is null from t limit 1；返回true select ‘null’is null from t limit 1;返回false |  |
| DISTINCT | | 支持 | 支持SELECT DISTINCT expr... 和 COUNT(DISTINCT expr...)等聚集函数 |  |
| INSERT | | 支持 | 支持INSERT到表或者目录中： INSERT [OVERWRITE] TABLE tablename1  select\_statement1 FROM from\_statement 和 INSERT OVERWRITE [LOCAL] DIRECTORY directory1  SELECT ... FROM ... |  |
| GROUP | | 支持 | GROUP BY expr,expr... |  |
| ON | | 支持 | a JOIN b ON(a.key=b.key) |  |
| COMMENT | | 支持 | 支持为列和表做注释 create table t（ a int comment 'a is int', b string comment 'b is string' ) comment 'this is a test table'; 在describe table的时候可以看到这些注释 |  |
| DESC | | 支持 | ORDER BY key ASC, value DESC 和 SORT BY key ASC, value DESC |  |
| LIKE | | 支持 |  |  |
| ASC | | 支持 | ORDER BY key ASC, value DESC 和 SORT BY key ASC, value DESC |  |
| ALTER | | 支持 | ALTER TABLE RENAME | ADDCOLS | PERPLACECOLS | ADDPARTITION... |  |
| SMALLINT | | 支持 | SMALLINT支持类型 |  |
| GRANT | | 支持 | GRANT privileges ON dbname.tablename TO username |  |
| DROP | | 支持 | drop table drop database |  |
| RENAME | | 支持 | ALTER TABLE RENAME |  |
| ORDER | | 有限支持 | ORDER BY key ASC, value DESC |  |
| INDEX | | 有限支持 | 仅支持索引的简单查询如: select t.key from t where t.value=10,这里t.value是索引字段 |  |
| COMPRESS | | 有限支持 | 只对结构化存储的文件进行压缩存储： create table zip(a int,b string) stored as formatfile compress |  |
| AUDIT | | 有限支持 | 只能通过TDW Server日志系统支持 |  |
| ROWNUM | | 不支持 | 不支持 |  |
| LEVEL | | 不支持 |  |
| WITH | | 支持 | 对with自动转换为subquery语法 |  |
| IN | | 变相支持 | 对于大部分IN，可以通过UDF实现， 带有SQL子句的IN，参考EXISTS改写 |  |
| BETWEEN | | 变相支持 | 可以通过UDF实现（正在开发） |  |
| EXISTS | | 变相支持 | SELECT a.key, a.value  FROM a  WHERE a.key in  (SELECT b.key  FROM B); 改写为：  SELECT a.key, a.val  FROM a LEFT SEMI JOIN b on (a.key = b.key) |  |
| ROLLUP | | 变相支持 | roll up(A,B,C)改写为：四个个group by语句，分别group by：（A），（A，B），（A，B，C）和一个全表的聚合 例如select t.a,t.b,t.c count(1) from t group by roll up(t.a,t.b,t.c)改写为： select t.a,count(1) from t group by t.a； select t.a,,t.b,count(1) from t group by t.a,t.b； select t.a,,t.b,t.c,count(1) from t group by t.a,t.b,t.c； select count(1) from t； |  |
| CUBE | | 变相支持 | cute(A,B)改写为：四个group by语句，分别group by：（A），（A，B），（B）和一个全表的聚合。 例如select t.a,t.b,t.c count(1) from t group by cube(t.a,t.b,t.c)改写为： select t.a,count(1) from t group by t.a； select t.a,,t.b,count(1) from t group by t.a,t.b； select t.a,,t.b,t.c,count(1) from t group by t.b； select count(1) from t； |  |
| DELETE | | 变相支持 | create-select-insert-rename-drop |  |
| UPDATE | | 变相支持 | create-select-insert-rename-drop |  |
| UNIQUE | | 变相支持 | ETL入库时进行去重 |  |
| CHAR | | 变相支持 | string |  |
| VARCHAR2 | | 变相支持 | string |  |
| NUMBER | | 变相支持 | int,bigint,tinyint,double,float |  |
| DATE | | 变相支持 | string和一系列string函数 |  |
| UNION | | 变相支持 | UNION ALL支持，UNION DISTINCT需要改写，改写为方法： select t1.key1,t1.value1 from t1 union distinct select t2.key2,t2.value2 from t2； 改写成：  select distinct tmp.k,tmp.v from  ( select key1 as k,value1 as v from t1  union all  select key2 as k,value2 as v from t2 ) tmp; |  |
| HAVING | | 支持 | HIVE>=0.7.0 |  |
| INTERSECT | | 变相支持 | select t1.key1,t1.value1 from t1 intersect select t2.key2,t2.value2 from t2; 改写成：  select tmp2.k,tmp2.v from (  select tmp.k as k,tmp.v as v ,count(1) as cnt from   (  select distinct t1.key1 as k,t1.value1 as v from t1   union all   select distinct t2.key2 as k,t2.value2 as v from t2   ) tmp   group by tmp.k,tmp.v ) tmp2  where tmp2.cnt =2 |  |
| VARCHAR | | 变相支持 | string |  |