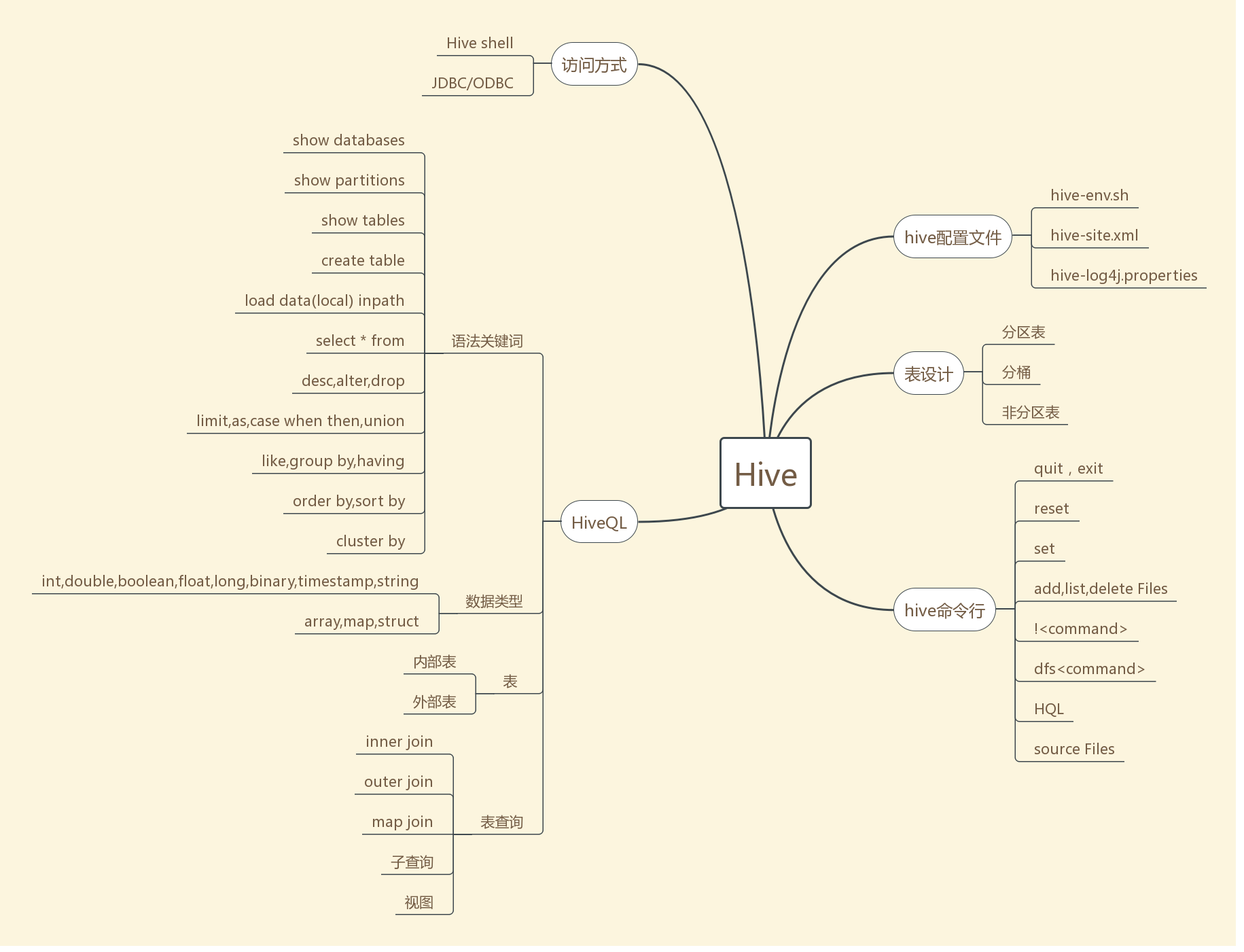
# Hive

## 12.1 概述

### 1.1 Hive简介

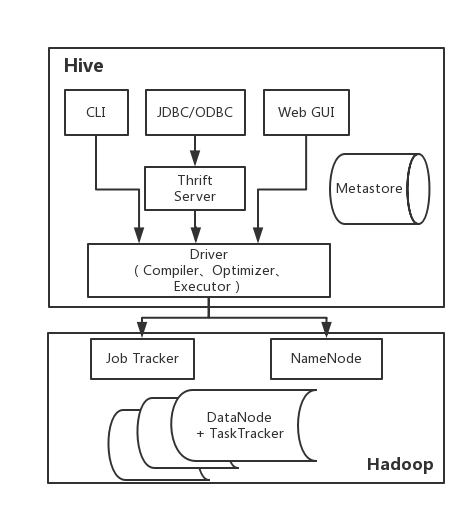
  Hive是Hadoop生态系统中必不可少的一个工具，最初由Fackbook提供，它提供了一种SQL（结构化查询语言）方言，可以查询存储在Hadoop分布式文件系统（HDFS）中的数据或其他Hadoop集成的文件系统，如MapR-FS、Amazon的S3以及像HBase和Cassandra这样的数据库中的数据。Hive设计的目的是让SQL技能良好，但Java技能较弱的分析师可以查询海量数据，2008年Fackbook把Hive项目贡献给Apache。Hive提供了比较完整的SQL功能，自身最大的缺点就是执行速度慢。

### 1.2 Hive知识体系



## 12.2 系统架构

### 2.1 Hive架构图



#### 2.1.1 用户接口

  Hive自带以下几种用户接口：

* CLI就是shell命令行；
* JDBC/ODBC是Hive的Java接口，与使用传统数据库JDBC的方式类似，通过Thrift访问MetaStore生成执行计划树；
* Web GUI HWI简单的网页界面，通过浏览器访问

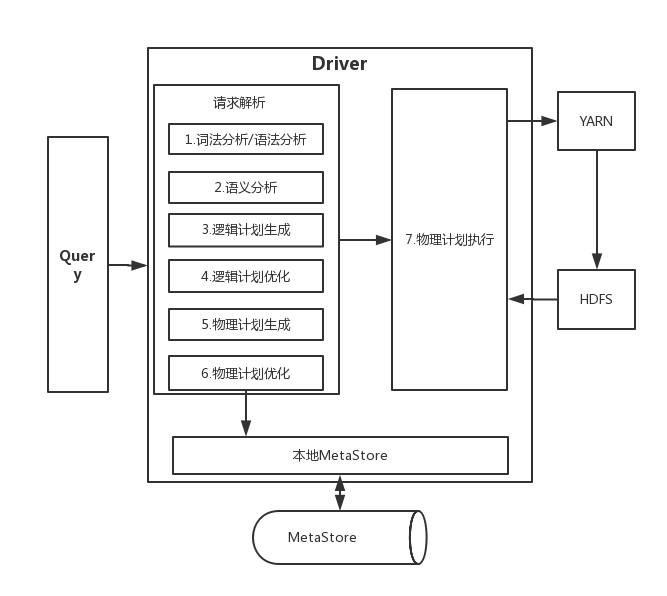
#### 2.1.2 命令执行

  所有的命令和查询都会进入到Driver，通过这个模块进行解析编译，对请求的计算进行优化，然后按照执行的步骤执行（通常是启动多个MapReduce任务-JOB来执行）。当需要启动MapReduce任务时，Hive本身不会生成Java MapReduce算法程序，相反，Hive通过一个表示‘JOB执行计划’的XML文件驱动执行内置的、原生的Mapper和Reducer模块。换句话说，这些通用的模块函数类似于微型的语言翻译程序，而驱动计算的‘语言’是以XML形式编码的。Hive通过和JobTracker通信来初始化MapReduce任务，而不必部署在JobTracker所在的管理节点上执行。要处理的数据文件是存储在HDFS中的，而HDFS是由NameNode进行管理的。

#### 2.1.3 元数据

  MetaStore默认使用Derby（单用户下使用），通常使用MySQL。Hive会在其中保存表模式和其他系统元数据。其中的元数据包括表信息，列信息，分区信息，以及表的数据所在的目录等。

### 2.2 Hive工作原理



  接收到一个SQL，接下来的处理流程如下：

* 1.词法分析/语法分析

  使用ANTLR将SQL语句解析成抽象语法树-AST；

* 2.语法分析

  从MetaStore获取模式信息，验证SQL语句中库名、表名、列名，以及数据类型的检查和隐式转换，以及Hive提供的函数和用户自定义的函数（UDF/UAF）；

* 3.逻辑计划生成

  生成逻辑计划-算子树；

* 4.逻辑计划优化

  对算子树进行优化，包括列剪枝，分区剪枝，谓词下推等；

* 5.物理计划生成

  将逻辑计划生成包含由MapReduce任务组成的DAG的物理计划；

* 6.物理计划优化

  对生成的MapReduce任务组成的DAG计划树进行优化，组合；

* 6.物理计划执行

  将DAG发送到Hadoop集群进行执行；

* 7.将查询结果返回

## 12.3 安装部署

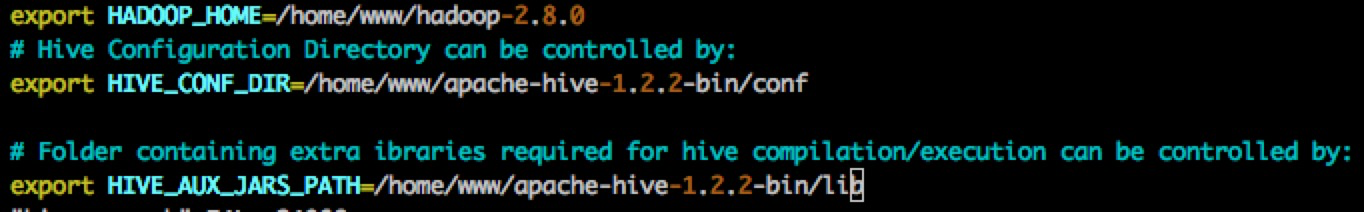
  Hive安装的前提是Java环境，Hadoop环境，MySQL安装完成。

### 3.1 安装Hive的版本及配置信息

* [Hive的版本 hive-1.2.2](http://mirrors.hust.edu.cn/apache/hive/hive-1.2.2/apache-hive-1.2.2-bin.tar.gz)
* hive的配置文件在$HIVE\_HOME/conf下，里面有5个默认的配置文件模板，特别注意还有两个位置的配置文件（如果没有指定$HIVE\_CONF\_DIR的话）/etc/hive/conf & /etc/hive/conf.dist
  + beenline-log4j.properties.template
  + hive-default.xml.template-> hive-site.xml
  + hive-env.sh.template->  [hive-env.sh](http://hive-env.sh)
  + hive-exec-log4j.properties.template
  + hive-log4j.properties.template-> hive-log4j.properties

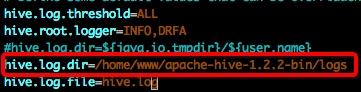
### 3.2 修改配置信息

#### 1.修改hive-env.sh



* 配置HADOOP\_HOME和HIVE\_CONF\_DIR

#### 2.修改hive-log4j.properties

* 首先mkdir logs创建存放日志文件的目录  
  kdir logs
* 配置  
  

#### 3.修改hive-site.xml

<property>

<name>javax.jdo.option.ConnectionURL</name>

<value> jdbc:mysql:*//localhost:3306/hivedb?createDatabaseIfNotExist=true</value>*

<describe>默认的配置元数据是存放在Derby数据库里面的，我们使用MySQL来存储元数据</describe>

</property>

<property>

<name>javax.jdo.option.ConnectionDriverName</name>

<value>com.mysql.jdbc.Driver</value>

</property>

<property>

<name>javax.jdo.option.ConnectionUserName</name>

<value>\*\*\*\*\*(根据自己的用户名填)</value>

</property>

<property>

<name>javax.jdo.option.ConnectionPassword</name>

<value>\*\*\*\*\*(根据自己的密码填)</value>

</property>

其他的一些配置项：

<property>

<name>hive.metastore.warehouse.dir</name>

<value>/opt/hive-1.2.1/warehouse</value>

</property>

<property>

<name>hive.exec.scratchdir</name>

<value>/opt/hive-1.2.1/scratchdir</value>

</property>

<property>

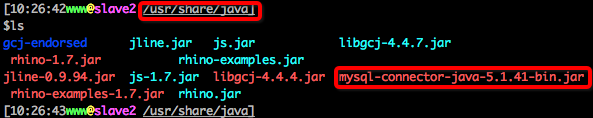
<name>hive.querylog.location</name>

<value>/opt/hive-1.2.1/logs</value>

</property>

* hive.metastore.warehouse.dir：指定hive的数据存储目录，指定的是HDFS上的位置，默认值:/usr/hive/warehouse，为了方便管理也可以在linux上指定一个目录。
* hive.exec.scratchdir：指定hive的临时数据目录，默认位置为：tmp/hive-${[user.name](http://user.name)}。为了方便管理也可以在linux上指定一个目录。

#### 4.上传JDBC jar包

  默认使用Derby数据库存放元数据，此时换成了MySQL作为数据库，需要上传MySQL的JDBC驱动包到$HIVE\_HOME/lib/。  
ysql connector jar  


  至此，hive安装部署完成，启动hive shell就可以开启hive的探索之旅了。

## 12.4 操作指南

### 4.1 基本操作

#### 4.1.1 介绍

* 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 EXISTS 选型来忽略这个异常。
* 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定的路径；若创建外部表，近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SOTORED 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，每一个分区单独存放在一个目录下。而且表和分区都可以对某个列进行CLUSTER BY操作，将若干个列放入到一个桶（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。  
  注：SerDe是Serialize/Deserialize的简称，目的是用于序列化和反序列化。

#### 4.1.2 语法

##### 1、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]

| **STORED** **BY** 'storage.handler.class.name' [ **WITH** SERDEPROPERTIES (...) ]

]

[LOCATION hdfs\_path]

[TBLPROPERTIES (property\_name=property\_value, ...)] [**AS** select\_statement] **CREATE** [**EXTERNAL**] **TABLE** [**IF** **NOT** **EXISTS**] table\_name

**LIKE** existing\_table\_name

[LOCATION hdfs\_path]

data\_type

: primitive\_type

| array\_type

| map\_type

| struct\_type

primitive\_type

: TINYINT

| SMALLINT

| INT

| BIGINT

| BOOLEAN

| FLOAT

| **DOUBLE**

| **STRING**

array\_type

: ARRAY < data\_type >

map\_type

: **MAP** < primitive\_type, data\_type >

struct\_type

: **STRUCT** < col\_name : data\_type [**COMMENT** col\_comment], ...>

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, ...)]

file\_format:

: SEQUENCEFILE

| TEXTFILE

| RCFILE (Note: **only** available **starting** **with** 0.6.0)

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

##### 2、添加分区

**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, ...)

##### 3、删除分区

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

##### 5、重命名表

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

##### 6、修改列/属性

**ALTER** **TABLE** table\_name **CHANGE** [**COLUMN**]

col\_old\_name col\_new\_name column\_type [**COMMENT** col\_comment] [**FIRST**|**AFTER** column\_name]

##### 7、添加/替换列

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

注：**add**是表示新增一字段，位置在所有列后面（**partition**列前）；**replace**则是表示替换表中所有字段。

##### 8、创建视图

**CREATE** **VIEW** [**IF** **NOT** **EXISTS**] view\_name [ (column\_name [**COMMENT** column\_comment], ...) ]

[**COMMENT** view\_comment]

[TBLPROPERTIES (property\_name = property\_value, ...)]

**AS** **SELECT** ...

注：视图是只读的，不能用**LOAD**/**INSERT**/**ALTER**。

##### 9、加载数据

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

##### 10、INSERT语法

Standard syntax:

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

Hive extension (multiple inserts):

**FROM** from\_statement

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

[**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

注：

\* 1 **insert**时，**from**子句既可以放在**select**子句后，也可以放在**insert**子句前。

\* 2 Hive不支持一条一条的用**insert**语句进行插入，也不支持**update**操作。数据是以**load**的方式，加载到建好的表中。数据一旦导入，则不可修改。要么**drop**掉整个表，要么建立新的表，导入新的数据。

##### 11 WRITE语法

Standard syntax:

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

Hive **extension** (**multiple inserts):**

FROM from\_statement

**INSERT** OVERWRITE [LOCAL] **DIRECTORY directory1** select\_statement1

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

##### 12 查询

**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**]

##### 13 Hive Join

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

注：Hive只支持等值连接（equality joins）、外连接（outer joins）和（left/right joins）。Hive不支持非等值的连接，因为非等值的连接很难转化为MapReduce任务。

另外，Hive支持多于两个表的连接。

#### 4.1.3 基本示例

* 1、如果一个表已经存在，可以使用IF NOT EXISTS；
* 2、建表示例

**create** **table** **user**

(

**id** int,

cont **string**

)

**row** **format** **delimited**

fileds **terminated** **by** '\005'

注：**terminated** **by** 关于来源的文本数据的字段间隔符。

* 3、如果要将自定义间隔符的文件读入一个表，需要通过创建表的语句来指明输入文件间隔符，然后load data 到这张表。
* 4、load data [local] inpath '/user/admin/a.txt' overwrite into table user;
* 5、使用EXTERNAL创建外表和普通创建管理表的区别是：前者存放元数据，删除后文件系统中的数据不会删除，后者会直接删除文件系统中的数据。
* 6.创建分区分桶表

**create** **table** **if** **not** **exists** page\_view

(

view\_time int,

user\_id bigint,

page\_url **string**,

referrer\_url **string**,

ip **string** **comment** 'ip address'

)**comment** 'page view table'

partitioned **by**(pt **string**,country **string**)

clustered **by**(user\_id) sorted **by**(view\_time) **into** 32 buckets

**row** **format** **delimited**

**fields** **terminated** **by** '\001'

collection items **terminated** **by** '\002'

**map** **keys** **terminated** **by** '\003'

**stored** **as** sequencefile;

* 7、复制一个空表

**create** **table** empty\_key\_value\_store **like** key\_value\_store;

### 4.2 函数以及UDF&UDAF&UDTF

#### 4.2.1 内置运算符

##### 4.2.1.1 关系运算符

| **运算符** | **类型** | **说明** |
| --- | --- | --- |
| A=B | 所有原始类型 | 如果A与B相等，则返回TRUE，否则返回FALSE。A == B，无效的语法，SQL使用‘=’，不使用‘==’。 |
| A<>B | 所有原始类型 | 如果A不等于B则返回TRUE，否则返回FALSE。如果A或B值为‘NULL’，结果返回‘NULL’。 |
| A<B | 所有原始类型 | 如果A小于B返回TRUE，否则返回FALSE。如果A或B为‘NULL’，结果返回‘NULL’。 |
| A<=B | 所有原始类型 | 如果A小于等于B返回TRUE，否则返回FALSE。如果A或B值为‘NULL’，结果返回‘NULL’。 |
| A>B | 所有原始类型 | 如果A大于B返回TRUE，否则返回FALSE。如果A或B为‘NULL’，结果返回‘NULL’。 |
| A>=B | 所有原始类型 | 如果A大于等于B返回TRUE，否则返回FALSE。如果A或B值为‘NULL’，结果返回‘NULL’。 |
| A is null | 所有类型 | 如果A值为‘NULL’，返回TRUE，否则返回FALSE。 |
| A is not null | 所有类型 | 如果A值不为‘NULL’，返回TRUE，否则返回FALSE。 |
| A like B | 字符串 | 如果A或B值为‘NULL’，结果返回‘NULL’。字符串A与B通过SQL进行匹配，如果相符返回TRUE，不符合返回FALSE。B字符串中的‘\_’代表任一字符，‘%’则代表任意多个字符。 |
| A rlike B | 字符串 | 如果A或B值为‘NULL’，结果返回‘NULL’。字符串A与B通过java正则进行匹配，如果相符返回TRUE，不符合返回FALSE。 |
| A regexp B | 字符串 | 与 rlike 相同。 |

##### 4.2.1.2 算术运算符

| **运算符** | **类型** | **说明** |
| --- | --- | --- |
| A+B | 所有数字类型 | A和B相加。结果与操作数值类型大的数据类型相同。 |
| A-B | 所有数字类型 | A和B相减。结果与操作数值有共同类型。 |
| A\*B | 所有数字类型 | A和B相乘，结果与操作数值有共同类型。需要说明的是，如果乘法造成溢出，将选择更高的类型。 |
| A/B | 所有数字类型 | A和B相除，结果是一个double（双精度）类型的结果。 |
| A%B | 所有数字类型 | A除以B的余数与操作数值有共同的类型。 |
| A&B | 所有数字类型 | 运算符查看两个参数的二进制表示法的值，并执行按位‘与’操作。两个表达式的一位均为1时，则结果的该位为1.否则，结果的该位为0. |
| A|B | 所有数字类型 | 运算符查看两个参数到的二进制表示法的值，并执行按位‘或’操作。只要任一表达式的一位为1，则结果的该位为1，否则，结果的该位为0。 |
| A^B | 所有数字类型 | 运算符查看两个参数的二进制表示法的值，并执行按位‘异或’操作。当且仅当一个表达式的某位上为1，结果的该位才为1.否则结果的该位为0. |
| ~A | 所有数字类型 | 对一个表达式执行按位‘非’（取反）。 |

##### 4.2.1.3 逻辑运算符

| **运算符** | **类型** | **说明** |
| --- | --- | --- |
| A and B | 布尔值 | A和B同时正确时，返回TRUE，否则返回FALSE。如果A或B值为NULL，返回NULL。 |
| A && B | 布尔值 | 与 A and B 相同。 |
| A or B | 布尔值 | A或B正确，或者两者同时正确，返回TRUE，否则返回FALSE。如果A和B同时为null，返回null。 |
| A | B | 布尔值 | 与 A or B 相同。 |
| not A | 布尔值 | 如果A为null或错误的时候返回TRUE，否则返回FALSE。 |
| !A | 布尔值 | 与 not A 相同。 |

##### 4.2.1.4 复杂类型函数

| **函数** | **类型** | **说明** |
| --- | --- | --- |
| map | (key1,value1,key2,value2,...) | 通过指定的键/值对，创建一个map。 |
| struct | (val1,val2,val3,...) | 通过指定的字段值，创建一个结构。 |
| array | (val1,val2,...) | 通过指定的元素，创建一个数组。 |

##### 4.2.1.5 对复杂类型函数操作

| **函数** | **类型** | **说明** |
| --- | --- | --- |
| A[n] | A是一个数组，n为int类型 | 返回数组A的第n个元素，第一个元素的索引为0. |
| M[key] | M是Map<k,v> | 返回关键词对应的值。 |
| S.x | S为struct | 返回x字符串在结构S中的存储位置。 |

#### 4.2.2 内置函数

##### 4.2.2.1 数学函数

| **返回类型** | **函数** | **说明** |
| --- | --- | --- |
| bigint | round(double a) | 四舍五入 |
| double | round(double a,int d) | 小数部分d位之后数字四舍五入 |
| bigint | floor(double a) | 对给定数据进行向下舍入，取最接近的整数。 |
| bigint | ceil(double a),ceiling(double a) | 对给定的数据向上舍入，取最近的整数。 |
| double | rand(),rand(int seed) | 返回大于或等于0且小于1的平均分布随机数。 |
| double | exp(double n) | 返回e的n次方 |
| double | ln(double a) | 返回给定数值的自然对数。 |
| double | log10(double a) | 返回给定数值的以10为底的自然对数。 |
| double | log10(double a) | 返回给定数值的以2为底的自然对数。 |
| double | log(double base,double a) | 返回给定底数及指数的自然对数。 |
| double | pow(double a,double p),power(double a,double p) | 返回某数的乘幂。 |
| double | sqrt(double a) | 返回数值的平方根。 |
| string | bin(bigint a) | 返回二进制格式。 |
| string | hex(bigint a) hex(string a) | 将整数或字符转换为十六进制格式。 |
| string | unhex(string a) | 十六进制字符转换为由数字表示的字符。 |
| string | conv(bigint num,int from\_base,int to\_base) | 将指定数值，由原来的度量体系转换为指定的度量体系。 |
| double | abs(double a) | 取绝对值。 |
| int,double | pmod(int a,int b),pmod(double a,double b) | 返回a除b的余数的绝对值。 |
| double | sin(double a) | 返回给定角度的正弦值。 |
| double | asin(double a) | 给出a的反正弦。 |
| double | cos(double a) | 返回余弦值。 |
| double | acos(double a) | 返回a的反余弦值。 |
| int,double | positive(int a),positive(double a) | 返回a的值。 |
| int,double | negative(int a),negative(double a) | 返回a的相反数。 |

##### 4.2.2.2 收集函数

| **返回类型** | **函数** | **说明** |
| --- | --- | --- |
| int | size(map<k,v>) | 返回map类型数据元素的数量。 |
| int | size(array<T>) | 返回数组类型的元素数量。 |

##### 4.2.2.3 类型转换函数

| **返回类型** | **函数** | **说明** |
| --- | --- | --- |
| z指定‘type’ | cast(expr as <type>) | 类型转换。 |

##### 4.2.2.4 日期函数

| **返回类型** | **函数** | **说明** |
| --- | --- | --- |
| string | from\_unixtime(bigint unixtime[,string format]) | UNIX\_TIMESTAMP参数表示返回一个值'YYY-MM-DD HH:mm:ss'或'YYYYmmddhhmmss.uuuuuu'格式，这取决于是否是在一个字符串或数字语境中使用的功能。该值表示当前的时区。 |
| bigint | unix\_timestamp() | 如果不带参数的调用，返回一个Unix时间戳(从1970-01-01 00:00:00到现在的UTC秒数)，为无符号整数。 |
| bigint | unix\_timestamp(string date) | 指定日期参数调用UNIX\_TIMESTAMP(),它返回从1970-01-01 00:00:00 到指定日期的秒数。 |
| bigint | unix\_timestamp(string date,string pattern) | 指定输入时间格式，返回从1970-01-01 00:00:00 到指定日期的秒数。 |
| string | to\_date(string timestamp) | 返回时间中的年月日。 |
| string | to\_dates(string date) | 给定一个日期date，返回一个天数（0年以来的天数）。 |
| int | year(string date) | 返回指定时间的年份。 |
| int | month(string date) | 返回指定时间的月份。 |
| int | day(string date),dayofmonth(date) | 返回指定时间的日期。 |
| int | hour(string date) | 返回指定时间的小时，范围为0到23. |
| int | minute(string date) | 返回指定时间的小时，范围为0到59. |
| int | second(string date) | 返回指定时间的秒，范围为0到59. |
| int | weekofyear(string) | 返回指定日期所在一年中的星期号，范围为0到53. |
| int | datediff(string enddate,string startdate) | 两个时间参数的日期之差。 |
| int | date\_add(string startdate,int days) | 给定时间，在此基础上加上指定的时间段。 |
| int | date\_sub(string startdate,int days) | 给定时间，在此基础上减去指定的时间段。 |

##### 4.2.2.5 条件函数

| **返回类型** | **函数** | **说明** |
| --- | --- | --- |
| T | if(boolean testCondition,T valueTrue,T valueFalseOrNull) | 判断是否满足条件，如果满足返回一个值，如果不满足则返回另一个值。 |
| T | coalesce(T v1,T v2,...) | 返回一组数据中，第一个不为null的值，如果均为null，返回null。 |
| T | case a when b then c [when d then e]\* [else f] end | 当a=b时，返回c；当a=d时，返回e，否则返回f。 |
| T | case when a then b [when c then d]\* [else e] end | 当值为a时返回b，当值为c时返回d，否则返回e。 |

##### 4.2.2.6 字符函数

| **返回类型** | **函数** | **说明** |
| --- | --- | --- |
| int | length(string a) | 返回字符串的长度。 |
| string | reverse(string a) | 返回倒序字符串。 |
| string | concat(string a,string b,...) | 连接多个字符串，合并为一个字符串，可以接受任意数量的输入字符串。 |
| string | concat\_ws(string sep,string a,string b,...) | 连接多个字符串，字符串之间以指定的分隔符分开。 |
| string | substr(string a,int start),substring(string a,int start) | 从文本字符串中指定起始位置后的字符。 |
| string | substr(string a,int start,int len),substring(string a,int start,int len) | 从文本字符串中指定位置指定长度的字符。 |
| string | upper(string a),ucase(string a) | 将文本字符串转换成字母全大写形式。 |
| string | lower(string a),lcase(string a) | 将文本字符串转换成字母全小写形式。 |
| string | trim(string a) | 删除字符串两端的空格，字符之间的空格保留。 |
| string | ltrim(string a) | 删除字符串左边的空格，其他的空格保留。 |
| string | rtrim(string a) | 删除字符串右边的空格，其他的空格保留。 |
| string | regexp\_replace(string a,string b,string c) | 字符串a中的b字符被c字符替代。 |
| string | regexp\_extract(string subject,string pattern,int index) | 通过下标返回正则表达式指定的部分。 |
| string | parse\_url(string url,string partextract[,string keyextract]) | 返回url中指定的部分。 |
| string | get\_json\_object(string json\_string,string path) |  |
| string | space(int n) | 返回指定数量的空格。 |
| string | repeat(string str,int n) | 重复n次字符串。 |
| int | ascii(string str) | 返回字符串中首字符的数字值。 |
| string | lpad(string str,int len,string pad) | 返回指定长度的字符串，给定字符串长度小于指定长度时，由指定字符从左侧填补。 |
| string | rpad(string str,int len,string pad) | 返回指定长度的字符串，给定字符串长度小于指定长度时，由指定字符从右侧填补。 |
| array | split(string str,string pattern) | 将字符串转换为数组。 |
| int | find\_in\_set(string str,string strList) | 返回字符串str第一次在strlist出现的位置。如果任一参数为null，返回null；如果第一个参数包含逗号，返回0. |
| array<array<string>>> | sentences(string str,string lang,string locale) | 将字符串内容按语句分组，每个单词间以逗号分隔，最后返回数组。 |
| array<struct<string,double>> | ngrams(array<array<string>>,int n,int k,int pf) |  |
| array<struct<string,double>> | context\_ngrams(array<array<string>>,int n,int k,int pf) |  |

#### 4.2.3 内置聚合函数(UDAF)

| **返回类型** | **函数** | **说明** |
| --- | --- | --- |
| bigint | count(\*),count(expr),count(distinct expr[,expr\_.,expr\_.]) | 返回记录条数。 |
| double | sum(col),sum(distinct col) | 求和。 |
| double | avg(col),avg(distinct col) | 求平均值。 |
| double | min(col) | 返回指定列中最小值。 |
| double | max(col) | 返回指定列中最大值。 |
| double | var\_pop(col) | 返回列的方差。 |
| double | var\_samp(col) | 返回指定列的样本方差。 |
| double | stddev\_pop(col) | 返回指定列的偏差。 |
| double | stddev\_samp(col) | 返回指定列的样本偏差。 |
| double | covar\_pop(col1,col2) | 两列数值协方差。 |
| double | covar\_samp(col1,col2) | 两列数值样本协方差。 |
| double | corr(col1,col2) | 返回两列数值的相关系数。 |
| double | percentile(col,p) | 返回数值区域的百分比数值点。0<=p<=1,否则返回null，不支持浮点型数值。 |
| array<double> | percentile(col,array(p1,[p2,...]) | 返回数值区域的一组百分比值分别对应的数值点。0<=p<=1,否则返回null，不支持浮点型数值。 |
| array | collect\_set(col) | 返回无重复记录。 |
| array | collect\_list(col) | 列转行。 |

#### 4.2.4 内置表生成函数(UDTF)

| **返回类型** | **函数** | **说明** |
| --- | --- | --- |
| array | explode(array<type> a) | 数组一条记录中有多个参数，将参数拆分，每个参数生成一列。 |
| json\_tuple() |  |  |

#### 4.2.5 用户自定义函数(UDF)

  Hive中，除了提供丰富的内置函数之外，还允许用户使用Java开发自定义的UDF函数。开发自定义UDF函数有两种方式，一种是继承org.apache.hadoop.hive.ql.exec.UDF，另一种是继承org.apache.hadoop.hive.ql.edf.generic.GenreicUDF.如果是针对简单的数据类型（比如String，Integer等）可以使用UDF，如果是针对复杂的数据类型（比如Array、Map、Struct等），可以使用GenericUDF，另外，GenericUDF还可以正在函数开始之前和结束之后做一些初始化和关闭的处理操作。

##### 4.2.5.1 UDF

  使用UDF非常简单，只需继承org.apache.hadoop.hive.ql.exec.UDF，并定义 public Object evaluate(Object args) {} 方法即可。 比如，下面的UDF函数实现了对一个String类型的字符串取HashMD5：

**import** org.apache.hadoop.hbase.util.**Bytes**;

**import** org.apache.hadoop.hbase.util.**MD5Hash**;

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

public **class** **HashMd5** **extends** **UDF** {

public **String** evaluate(**String** cookie) {

**return** **MD5Hash**.getMD5AsHex(**Bytes**.toBytes(cookie));

}

}

将上面的HashMd5类打成jar包，udf.jar  
使用时候，在Hive命令行执行：

add jar file:///tmp/udf.jar;

**CREATE** **temporary** **function** str\_md5 **as** 'com.lxw1234.hive.udf.HashMd5';

**select** str\_md5(‘lxw1234.com’) **from** dual;

##### 4.2.5.2 GenericUDF

  继承org.apache.hadoop.hive.ql.udf.generic.GenericUDF之后，需要重写几个重要的方法：

###### 1.public void configure(MapredContext context){}

可选，该方法中可以通过context.getJobConf()获取job执行时候的Configuration；可以通过Configuration传递参数值。

###### 2.public ObjectInspector initialize(ObjectINspector[] arguments)

必选，该方法用于函数初始化操作，并定义函数的返回值类型；比如，在该方法中可以初始化对象实例，初始化数据库连接，初始化读取文件等。

###### 3.public Object evaluate(DeferredObject[] args){}

必选，函数处理的核心方法，用途和UDF中的evaluate一样。

###### 4.public String getDisplayString(String[] children)

必选，显示函数的帮助信息。

###### 5.public void close()

可选，map完成后，执行关闭操作。

###### 下面的程序将一个以逗号分隔的字符串，切分成List，并返回：

**import** java.util.ArrayList;

**import** java.util.Date;

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

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

**import** org.apache.hadoop.hive.ql.metadata.HiveException;

**import** org.apache.hadoop.hive.ql.udf.generic.GenericUDF;

**import** org.apache.hadoop.hive.serde2.objectinspector.ObjectInspector;

**import** org.apache.hadoop.hive.serde2.objectinspector.ObjectInspectorFactory;

**import** org.apache.hadoop.hive.serde2.objectinspector.PrimitiveObjectInspector;

**import** org.apache.hadoop.hive.serde2.objectinspector.primitive.PrimitiveObjectInspectorFactory;

*/\*\**

*\* 该函数用于将字符串切分成List，并返回*

*\*/*

**public** class CheckGenericUDF extends GenericUDF {

**private** **static** int mapTasks = 0;

**private** **static** **String** init = "";

**private** **transient** ArrayList ret = **new** ArrayList();

@Override

**public** **void** configure(MapredContext context) {

System.out.println(**new** Date() + "######## configure");

**if**(**null** != context) {

*//从jobConf中获取map数*

mapTasks = context.getJobConf().getNumMapTasks();

}

System.out.println(**new** Date() + "######## mapTasks [" + mapTasks + "] ..");

}

@Override

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

**throws** UDFArgumentException {

System.out.println(**new** Date() + "######## initialize");

*//初始化文件系统，可以在这里初始化读取文件等*

init = "init";

*//定义函数的返回类型为java的List*

ObjectInspector returnOI = PrimitiveObjectInspectorFactory

.getPrimitiveJavaObjectInspector(PrimitiveObjectInspector.PrimitiveCategory.STRING);

**return** ObjectInspectorFactory.getStandardListObjectInspector(returnOI);

}

@Override

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

ret.clear();

**if**(args.length < 1) **return** ret;

*//获取第一个参数*

**String** str = args[0].get().toString();

**String**[] s = str.split(",",-1);

**for**(**String** word : s) {

ret.add(word);

}

**return** ret;

}

@Override

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

**return** "Usage: Lxw1234GenericUDF(String str)";

}

}

在MapReduce阶段，GenericUDF几个方法的执行顺序为：configure->initialize->evaluate->close。

### 4.3 Hive正则

Hive的正则语法和Java的正则语法一致。

#### 正则表达式语法

在其他语言中，\\表示：我想要在正则表达式中插入一个普通的反斜杠，请不要给他任何特殊的意义。  
在Java中，\\表示：我要插入一个正则表达式的反斜杠，所以其后的字符具有特殊的意义。 所以，在其他的语言中（如Perl），一个反斜杠\就足以具有转义的作用，而在Java正则表达式中则需要有两个反斜杠才能被解析为其他语言中的转义作用。

| **字符** | **说明** |
| --- | --- |
| \ | 将下一字符标记为特殊字符、文本、反向引用或八进制转义符。例如，‘n’匹配字符n；'\n'匹配换行符；序列'\\'匹配,'\('匹配'('。 |
| ^ | 匹配输入字符串开始的位置。 |
| $ | 匹配输入字符串结尾的位置。 |
| \* | 零次或多次匹配前面的字符或子表达式。 |
| + | 一次或多次匹配前面的字符或子表达式。 |
| ? | 零次或一次匹配前面的字符或子表达式。 |
| {n} | n是非负整数。正好匹配n次。 |
| {n,} | n是非负整数整数。至少匹配n次。 |
| {n,m} | M 和 n 是非负整数，其中 n <= m。匹配至少 n 次，至多 m 次。例如，"o{1,3}"匹配"fooooood"中的头三个 o。'o{0,1}' 等效于 'o?'。注意：您不能将空格插入逗号和数字之间。 |
| ? | 当此字符紧随任何其他限定符（\*、+、?、{n}、{n,}、{n,m}）之后时，匹配模式是"非贪心的"。"非贪心的"模式匹配搜索到的、尽可能短的字符串，而默认的"贪心的"模式匹配搜索到的、尽可能长的字符串。例如，在字符串"oooo"中，"o+?"只匹配单个"o"，而"o+"匹配所有"o"。 |
| . | 匹配除"\r\n"之外的任何单个字符。若要匹配包括"\r\n"在内的任意字符，请使用诸如"[\s\S]"之类的模式。 |
| (pattern) | 匹配pattern并捕获该匹配的子表达式。可以使用$0...$9属性从结果匹配集合中检索捕获的匹配。 |
| (?:pattern) | 匹配 pattern 但不捕获该匹配的子表达式，即它是一个非捕获匹配，不存储供以后使用的匹配。这对于用"or"字符 ( |
| (?=pattern) | 执行正向预测先行搜索的子表达式，该表达式匹配处于匹配 pattern 的字符串的起始点的字符串。它是一个非捕获匹配，即不能捕获供以后使用的匹配。例如，'Windows (?=95 |
| (?!pattern) | 执行反向预测先行搜索的子表达式，该表达式匹配不处于匹配 pattern 的字符串的起始点的搜索字符串。它是一个非捕获匹配，即不能捕获供以后使用的匹配。例如，'Windows (?!95 |
| x|y | 匹配 x 或 y。例如，'z |
| [xyz] | 字符集。匹配包含的任一字符。例如，"[abc]"匹配"plain"中的"a"。 |
| [^xyz] | 反向字符集。匹配未包含的任何字符。例如，"[^abc]"匹配"plain"中"p"，"l"，"i"，"n"。 |
| [a-z] | 字符范围。匹配指定范围内的任何字符。例如，"[a-z]"匹配"a"到"z"范围内的任何小写字母。 |
| [^a-z] | 反向范围字符。匹配不在指定的范围内的任何字符。例如，"[^a-z]"匹配任何不在"a"到"z"范围内的任何字符。 |
| \b | 匹配一个字边界，即字与空格间的位置。例如，"er\b"匹配"never"中的"er"，但不匹配"verb"中的"er"。 |
| \B | 非字边界匹配。"er\B"匹配"verb"中的"er"，但不匹配"never"中的"er"。 |
| \cx | 匹配 x 指示的控制字符。例如，\cM 匹配 Control-M 或回车符。x 的值必须在 A-Z 或 a-z 之间。如果不是这样，则假定 c 就是"c"字符本身。 |
| \d | 数字字符匹配。等效于 [0-9]。 |
| \D | 非数字字符匹配。等效于 [^0-9]。 |
| \f | 换页符匹配。等效于 \x0c 和 \cL。 |
| \n | 换行符匹配。等效于 \x0a 和 \cJ。 |
| \r | 匹配一个回车符。等效于 \x0d 和 \cM。 |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等。与 [ \f\n\r\t\v] 等效。 |
| \S | 匹配任何非空白字符。与 [^ \f\n\r\t\v] 等效。 |
| \t | 制表符匹配。与 \x09 和 \cI 等效。 |
| \v | 垂直制表符匹配。与 \x0b 和 \cK 等效。 |
| \w | 匹配任何字类字符，包括下划线。与"[A-Za-z0-9\_]"等效。 |
| \W | 与任何非单词字符匹配。与"[^A-Za-z0-9\_]"等效。 |
| \xn | 匹配 n，此处的 n 是一个十六进制转义码。十六进制转义码必须正好是两位数长。例如，"\x41"匹配"A"。"\x041"与"\x04"&"1"等效。允许在正则表达式中使用 ASCII 代码。 |
| \num | 匹配 num，此处的 num 是一个正整数。到捕获匹配的反向引用。例如，"(.)\1"匹配两个连续的相同字符。 |
| \n | 标识一个八进制转义码或反向引用。如果 \n 前面至少有 n 个捕获子表达式，那么 n 是反向引用。否则，如果 n 是八进制数 (0-7)，那么 n 是八进制转义码。 |
| \nm | 标识一个八进制转义码或反向引用。如果 \nm 前面至少有 nm 个捕获子表达式，那么 nm 是反向引用。如果 \nm 前面至少有 n 个捕获，则 n 是反向引用，后面跟有字符 m。如果两种前面的情况都不存在，则 \nm 匹配八进制值 nm，其中 n 和 m 是八进制数字 (0-7)。 |
| \nml | 当 n 是八进制数 (0-3)，m 和 l 是八进制数 (0-7) 时，匹配八进制转义码 nml。 |
| \un | 匹配 n，其中 n 是以四位十六进制数表示的 Unicode 字符。 |

### 4.4 Hive优化

  Hive通过将查询划分成一个或多个MapReduce任务达到并行处理的目的。每个任务都可能具有多个mapper和reducer任务，其中至少有一些是可以并行执行的。确定最佳的mapper个数和reducer个数取决于多个变量，例如输入的数据量大小以及对这些数据执行的操作类型等。保持平衡性是很有必要的，对于Spark/Hadoop这样的大数据系统来讲，数据量大并不可怕，可怕的是数据倾斜，每个节点处理的运算不均衡。如果有太多的mapper或reducer任务，就会导致启动阶段、调度和运行job过程中产生过多的开销；而如果设置的数量太少，那就有可能没充分利用好集群内在并行性。

**4.4.1 配置列表**

|  |  |  |
| --- | --- | --- |
| **配置项** | **默认值** | **说明** |
| mapred.reduce.tasks | 1 | 所提交Job的reducer的个数，使用Hadoop Client的配置。 |
| hive.mapred.mode | nonstrict | Map/Reduce模式，如果设置为strict，将禁止三种类型的查询： 1. 分区表的where筛选条件必须含有分区字段； 2.对于使用了order by语句的查询，必须使用limit语句（order by语句为执行排序会将所有的结果集数据分发到同一个reducer中进行处理，增加limit语句可以防止reducer额外执行很长时间。） 3.限制笛卡尔积的查询，就是有where语句，而没有on语句。 |
| hive.merge.mapfiles | true | 在Map-only的任务结束时合并小文件，是否开启合并Map端小文件，当Hive输入由很多小文件组成，由于每个小文件都会启动一个map任务，如果文件过小，会使得map任务启动和初始化的时间大于逻辑处理的时间，造成资源浪费，甚至OOM。为此，当我们启动一个任务，发现输入数据量小但是任务数量多时，需要注意在Map端进行输入合并。当然，我们向一个表写数据时，也需要注意输出文件的大小。 |
| hive.merge.mapredfiles | false | 是否开启合并 Map/Reduce 小文件，即是否在 Map-Reduce 的任务结束时合并小文件。 |
| hive.exec.parallel | false | 是否开启 map/reduce job 的并发提交。 |
| hive.limit.optimize.enable | false | 当使用limit语句时，其可以对数据源进行抽样，避免执行整个查询语句，然后再返回部分结果。这个功能有个缺点：有可能输入中有用的数据永远不会被处理到。 |
| hive.exec.reducers.bytes.per.reducer | 1000000000 | 每一个reducer的平均负载字节数。 |
| hive.exec.reducers.max | 999 | 设置reducer个数的上限，可以阻止某个查询消耗过多的reducer资源，对于这个属性值大小的设定，一个建议的计算公式如下： (集群总reducer槽位个数\*1.5)/(执行中查询的平均个数) 1.5倍是一个经验系数，用于防止未充分利用集群的情况。 |
| hive.exec.rowoffset | true | hive提供了2种虚拟列：一种用于将要进行划分的输入文件名；另一种用于文件中的块内偏移量。当hive产生了非预期的或null的返回结果时，可以通过这些虚拟列诊断查询。通过这些“字段”，用户可以查看到哪个文件甚至哪些数据导致出现问题： SELECT INPUT\_FILE\_NAME,BLOCK\_OFFSET\_INSIDE\_FILE,ROW\_OFFSET\_INSIDE\_BLOCK,line from hive\_text where line like '%hive%' limit 2 |
| hive.multigroupby.singlemr | false | 一个特别的优化，是否将查询中的多个group by操作组装到单个MapReduce任务中。 |
| hive.exec.dynamic.partition | false | 是否打开动态分区。 |
| hive.exec.dynamic.partition.mode | strict | 打开动态分区后，动态分区的模式，有strict和nonstrict两个值可选，strict要求至少包含一个静态分区列，nonstrict则无此要求。 |
| hive.exec.max.dynamic.partitions | 1000 | 所允许的最大的动态分区的个数。 |
| hive.exec.max.dynamic.partitions.pernode | 100 | 单个reduce节点所允许的最大的动态分区的个数。 |
| hive.exec.default.partition.name | \_HIVE\_DEFAULT\_PARTITION\_ | 默认的动态分区的名称，当动态分区列为‘’或为null时，使用此名称。 |
| hive.exec.mode.local.auto | true | 决定Hive是否应该自动地根据输入文件大小，在本地运行（在GateWay运行）。 |
| hive.exec.mode.local.auto.inputbytes.max | 134217728L | 如果hive.exec.mode.local.auto为true，当输入文件大小小于此阈值时可以自动在本地模式运行，默认是128M。 |
| hive.exec.mode.local.auto.tasks.max | 4 | 如果hive.exec.mode.local.auto为true，则Hive Task（Hadoop Jobs）小于此阈值时，可以自动在本地模式运行。 |
| hive.auto.convert.join | false | 是否根据输入小表的大小，自动将Reduce端的Common Join转化为Map Join，从而加快大表关联小表的Join速度。 |
| hive.mapred.local.mem | 0 | Mapper/Reducer在本地模式的最大内存量，以字节为单位，0为不限制。 |
| hive.exec.scratchdir | /tmp/<user.name>/hive | HDFS路径，用于存储不同map/reduce阶段的执行计划和这些阶段的中间输出结果。 |
| hive.metastore.warehouse.dir | ” | Hive默认的数据文件存储路径，通常为HDFS可写的路径。 |
| hive.groupby.skewindata | false | 决定group by操作是否支持倾斜的数据。 |
| hive.map.aggr | true | 决定是否可以在Map端进行聚合操作。 |
| hive.join.emit.interval | 1000 | Hive join操作的发射时间间隔，以毫秒为单位。 |
| hive.mapjoin.cache.numrows | 25000 | Hive Map Join所缓存的行数。 |
| hive.groupby.mapaggr.checkinterval | 100000 | 对于Group By操作的Map聚合的检测时间，以毫秒为单位。 |
| hive.map.aggr.hash.percentmemory | 0.5 | Hive Map端聚合的哈希存储所占用虚拟机的内存比例。 |
| hive.map.aggr.hash.min.reduction | 0.5 | Hive Map聚合的哈希存储的最小reduce比例。 |
| hive.merge.smallfiles.avgsize | 16000000 | 需要合并的小文件群的平均大小，默认16M。 |
| hive.merge.size.per.task | 256000000 | 每个任务合并后文件的大小，根据此大小确定reducer的个数，默认256M。 |
| mapred.min.split.size | 1 | Map Reduce Job的最小输入切分大小，与Hadoop Client使用相同的配置。 |
| hive.mergejob.maponly | true | 是否启用Map Only的合并Job。 |
| hive.mapjoin.maxsize | 1000000 | Map Join所处理的最大行数。超过此行数，Map Join进程会异常退出。 |
| hive.optimize.groupby | true | 是否优化group by。 |
| hive.optimize.bucketmapjoin | false | 是否优化bucket map join。 |