**Hive实战使用指南**

**VESION 1.0**

创建时间：2016年09月

目录

[1 绪论 7](#_Toc464056142)

[1.1 编写背景 7](#_Toc464056143)

[1.2 文档说明 7](#_Toc464056144)

[2 Hive简介 7](#_Toc464056145)

[2.1 Hive定义 7](#_Toc464056146)

[3 Hive体系架构 8](#_Toc464056147)

[3.1 用户接口 8](#_Toc464056148)

[3.2 元数据存储 8](#_Toc464056149)

[3.3 Hadoop 8](#_Toc464056150)

[3.4 Hive 的数据存储 8](#_Toc464056151)

[4 Hive数据类型 8](#_Toc464056152)

[4.1 基本数据类型 8](#_Toc464056153)

[4.2 复杂数据类型 9](#_Toc464056154)

[4.3 数据类型建议 10](#_Toc464056155)

[5 Hive存储格式 10](#_Toc464056156)

[5.1 TEXTFIEL 10](#_Toc464056157)

[5.2 SEQUENCEFILE 10](#_Toc464056158)

[5.3 RCFILE 11](#_Toc464056159)

[5.4 自定义格式 11](#_Toc464056160)

[5.5 存储格式建议 11](#_Toc464056161)

[6 Hive编解码器(压缩格式) 11](#_Toc464056162)

[6.1 压缩格式对比 12](#_Toc464056163)

[6.2 压缩格式建议 12](#_Toc464056164)

[7 HiveQL:数据定义 12](#_Toc464056165)

[7.1 Hive中的数据库 12](#_Toc464056166)

[7.1.1 创建数据库 13](#_Toc464056167)

[7.1.2 查看数据库 13](#_Toc464056168)

[7.1.3 删除数据库 13](#_Toc464056169)

[7.1.4 进入数据库 14](#_Toc464056170)

[7.2 Hive的表 14](#_Toc464056171)

[7.2.1 管理表 14](#_Toc464056172)

[7.2.2 查看表 14](#_Toc464056173)

[7.2.3 外部表 15](#_Toc464056174)

[7.2.4 管理分区表 15](#_Toc464056175)

[7.2.5 外部分区表 16](#_Toc464056176)

[7.2.6 删除表 16](#_Toc464056177)

[7.2.7 表重命名 17](#_Toc464056178)

[7.2.8 增加、修改、和删除表分区 17](#_Toc464056179)

[8 HiveQL:数据操作 19](#_Toc464056180)

[8.1 向管理表中装载数据 19](#_Toc464056181)

[8.2 通过查询语句向表中插入数据 19](#_Toc464056182)

[8.3 导出数据 20](#_Toc464056183)

[8.4 导出数据建议 21](#_Toc464056184)

[9 HiveQL:查询 21](#_Toc464056185)

[9.1 避免Mapruduce查询 21](#_Toc464056186)

[9.2 Join语句 23](#_Toc464056187)

[9.2.1 INNER JOIN 23](#_Toc464056188)

[9.2.2 LEFT OUTER JOIN 23](#_Toc464056189)

[9.2.3 RIGHT OUTER JOIN 23](#_Toc464056190)

[9.2.4 FULL OUTER JOIN 23](#_Toc464056191)

[9.2.5 LEFT SEMI JOIN 23](#_Toc464056192)

[9.2.6 MAP SIDE JOIN 23](#_Toc464056193)

[9.3 Order by, Sort by ,Dristribute by,Cluster By 24](#_Toc464056194)

[9.3.1 order by 24](#_Toc464056195)

[9.3.2 sort by 24](#_Toc464056196)

[9.3.3 distribute by 24](#_Toc464056197)

[9.3.4 Cluster By 25](#_Toc464056198)

[10 HiveQL:视图 25](#_Toc464056199)

[10.1 创建View 25](#_Toc464056200)

[10.2 删除view 25](#_Toc464056201)

[10.3 修改view 25](#_Toc464056202)

[11 HiveQL:索引 26](#_Toc464056203)

[11.1 创建索引： 26](#_Toc464056204)

[11.2 重建索引： 27](#_Toc464056205)

[11.3 删除索引 27](#_Toc464056206)

[12 Hive元数据 27](#_Toc464056207)

[12.1 数据字典 27](#_Toc464056208)

[13 数据倾斜 28](#_Toc464056209)

[13.1 数据倾斜的原因 28](#_Toc464056210)

[13.1.1 操作 28](#_Toc464056211)

[13.1.2 原因 28](#_Toc464056212)

[13.1.3 表现 28](#_Toc464056213)

[13.2 数据倾斜的解决方案 29](#_Toc464056214)

[13.2.1 参数调节 29](#_Toc464056215)

[13.2.2 SQL语句调节 29](#_Toc464056216)

[13.2.3 空值产生的数据倾斜 29](#_Toc464056217)

[13.3 不同数据类型关联产生数据倾斜 30](#_Toc464056218)

[13.3.1 小表不小不大，怎么用 map join 解决倾斜问题 30](#_Toc464056219)

[13.4 总结 31](#_Toc464056220)

[14 Hive参数优化 31](#_Toc464056221)

[14.1 本地模式(小任务) 31](#_Toc464056222)

[14.2 并发执行 31](#_Toc464056223)

[14.3 Strict Mode 32](#_Toc464056224)

[14.4 动态分区： 32](#_Toc464056225)

[14.5 推测执行： 32](#_Toc464056226)

[14.6 Single MapReduce MultiGROUP BY 32](#_Toc464056227)

[14.7 是否提供虚拟列 32](#_Toc464056228)

[14.8 分组 32](#_Toc464056229)

[14.9 Map端部分聚合 33](#_Toc464056230)

[14.10 Multi- Group-By Inserts 33](#_Toc464056231)

[14.11 排序 33](#_Toc464056232)

[14.12 合并小文件 33](#_Toc464056233)

[14.13 Map/reduce数目 34](#_Toc464056234)

[15 常用优化方向 34](#_Toc464056235)

[16 HiveQL:函数 35](#_Toc464056236)

[16.1 关系运算： 35](#_Toc464056237)

[16.1.1 等值比较: = 35](#_Toc464056238)

[16.1.2 不等值比较: <> 35](#_Toc464056239)

[16.1.3 小于比较: < 35](#_Toc464056240)

[16.1.4 小于等于比较: <= 35](#_Toc464056241)

[16.1.5 大于比较: > 37](#_Toc464056242)

[16.1.6 大于等于比较: >= 37](#_Toc464056243)

[16.1.7 空值判断: IS NULL 37](#_Toc464056244)

[16.1.8 非空判断: IS NOT NULL 37](#_Toc464056245)

[16.1.9 LIKE比较: LIKE 38](#_Toc464056246)

[16.1.10 JAVA的LIKE操作: RLIKE 38](#_Toc464056247)

[16.1.11 REGEXP操作: REGEXP 38](#_Toc464056248)

[16.2 数学运算： 39](#_Toc464056249)

[16.2.1 加法操作: + 39](#_Toc464056250)

[16.2.2 减法操作: - 39](#_Toc464056251)

[16.2.3 乘法操作: \* 39](#_Toc464056252)

[16.2.4 除法操作: / 40](#_Toc464056253)

[16.2.5 取余操作: % 40](#_Toc464056254)

[16.2.6 位与操作: & 41](#_Toc464056255)

[16.2.7 位或操作: | 41](#_Toc464056256)

[16.2.8 位异或操作: ^ 41](#_Toc464056257)

[16.2.9 位取反操作: ~ 41](#_Toc464056258)

[16.3 逻辑运算： 42](#_Toc464056259)

[16.3.1 逻辑与操作: AND 42](#_Toc464056260)

[16.3.2 逻辑或操作: OR 42](#_Toc464056261)

[16.3.3 逻辑非操作: NOT 42](#_Toc464056262)

[16.4 数值计算 43](#_Toc464056263)

[16.4.1 取整函数: round 43](#_Toc464056264)

[16.4.2 指定精度取整函数: round 43](#_Toc464056265)

[16.4.3 向下取整函数: floor 43](#_Toc464056266)

[16.4.4 向上取整函数: ceil 43](#_Toc464056267)

[16.4.5 向上取整函数: ceiling 44](#_Toc464056268)

[16.4.6 取随机数函数: rand 44](#_Toc464056269)

[16.4.7 自然指数函数: exp 44](#_Toc464056270)

[16.4.8 以10为底对数函数: log10 45](#_Toc464056271)

[16.4.9 以2为底对数函数: log2 45](#_Toc464056272)

[16.4.10 对数函数: log 45](#_Toc464056273)

[16.4.11 幂运算函数: pow 46](#_Toc464056274)

[16.4.12 幂运算函数: power 46](#_Toc464056275)

[16.4.13 开平方函数: sqrt 46](#_Toc464056276)

[16.4.14 二进制函数: bin 46](#_Toc464056277)

[16.4.15 十六进制函数: hex 47](#_Toc464056278)

[16.4.16 反转十六进制函数: unhex 47](#_Toc464056279)

[16.4.17 进制转换函数: conv 47](#_Toc464056280)

[16.4.18 绝对值函数: abs 47](#_Toc464056281)

[16.4.19 正取余函数: pmod 48](#_Toc464056282)

[16.4.20 正弦函数: sin 48](#_Toc464056283)

[16.4.21 反正弦函数: asin 48](#_Toc464056284)

[16.4.22 余弦函数: cos 49](#_Toc464056285)

[16.4.23 反余弦函数: acos 49](#_Toc464056286)

[16.4.24 positive函数: positive 49](#_Toc464056287)

[16.4.25 negative函数: negative 49](#_Toc464056288)

[16.5 日期函数 50](#_Toc464056289)

[16.5.1 UNIX时间戳转日期函数: from\_unixtime 50](#_Toc464056290)

[16.5.2 获取当前UNIX时间戳函数: unix\_timestamp 50](#_Toc464056291)

[16.5.3 日期转UNIX时间戳函数: unix\_timestamp 50](#_Toc464056292)

[16.5.4 指定格式日期转UNIX时间戳函数: unix\_timestamp 50](#_Toc464056293)

[16.5.5 日期时间转日期函数: to\_date 51](#_Toc464056294)

[16.5.6 日期转年函数: year 51](#_Toc464056295)

[16.5.7 日期转月函数: month 51](#_Toc464056296)

[16.5.8 日期转天函数: day 51](#_Toc464056297)

[16.5.9 日期转小时函数: hour 52](#_Toc464056298)

[16.5.10 日期转分钟函数: minute 52](#_Toc464056299)

[16.5.11 日期转秒函数: second 52](#_Toc464056300)

[16.5.12 日期转周函数: weekofyear 52](#_Toc464056301)

[16.5.13 日期比较函数: datediff 53](#_Toc464056302)

[16.5.14 日期增加函数: date\_add 53](#_Toc464056303)

[16.5.15 日期减少函数: date\_sub 53](#_Toc464056304)

[16.6 条件函数 53](#_Toc464056305)

[16.6.1 If函数: if 53](#_Toc464056306)

[16.6.2 非空查找函数: COALESCE 54](#_Toc464056307)

[16.6.3 条件判断函数：CASE 54](#_Toc464056308)

[16.6.4 条件判断函数：CASE 54](#_Toc464056309)

[16.7 字符串函数 55](#_Toc464056310)

[16.7.1 字符串长度函数：length 55](#_Toc464056311)

[16.7.2 字符串反转函数：reverse 55](#_Toc464056312)

[16.7.3 字符串连接函数：concat 55](#_Toc464056313)

[16.7.4 带分隔符字符串连接函数：concat\_ws 55](#_Toc464056314)

[16.7.5 字符串截取函数：substr,substring 56](#_Toc464056315)

[16.7.6 字符串截取函数：substr,substring 56](#_Toc464056316)

[16.7.7 字符串转大写函数：upper,ucase 56](#_Toc464056317)

[16.7.8 字符串转小写函数：lower,lcase 56](#_Toc464056318)

[16.7.9 去空格函数：trim 57](#_Toc464056319)

[16.7.10 左边去空格函数：ltrim 57](#_Toc464056320)

[16.7.11 右边去空格函数：rtrim 57](#_Toc464056321)

[16.7.12 正则表达式替换函数：regexp\_replace 57](#_Toc464056322)

[16.7.13 正则表达式解析函数：regexp\_extract 58](#_Toc464056323)

[16.7.14 URL解析函数：parse\_url 58](#_Toc464056324)

[16.7.15 json解析函数：get\_json\_object 59](#_Toc464056325)

[16.7.16 空格字符串函数：space 59](#_Toc464056326)

[16.7.17 重复字符串函数：repeat 59](#_Toc464056327)

[16.7.18 首字符ascii函数：ascii 60](#_Toc464056328)

[16.7.19 左补足函数：lpad 60](#_Toc464056329)

[16.7.20 右补足函数：rpad 60](#_Toc464056330)

[16.7.21 分割字符串函数: split 60](#_Toc464056331)

[16.7.22 集合查找函数: find\_in\_set 60](#_Toc464056332)

[16.8 合统计函数 61](#_Toc464056333)

[16.8.1 个数统计函数: count 61](#_Toc464056334)

[16.8.2 总和统计函数: sum 61](#_Toc464056335)

[16.8.3 平均值统计函数: avg 61](#_Toc464056336)

[16.8.4 最小值统计函数: min 62](#_Toc464056337)

[16.8.5 最大值统计函数: max 62](#_Toc464056338)

[16.8.6 非空集合总体变量函数: var\_pop 62](#_Toc464056339)

[16.8.7 非空集合样本变量函数: var\_samp 62](#_Toc464056340)

[16.8.8 总体标准偏离函数: stddev\_pop 62](#_Toc464056341)

[16.8.9 样本标准偏离函数: stddev\_samp 63](#_Toc464056342)

[16.8.10 中位数函数: percentile 63](#_Toc464056343)

[16.8.11 中位数函数: percentile 63](#_Toc464056344)

[16.8.12 近似中位数函数: percentile\_approx 64](#_Toc464056345)

[16.8.13 近似中位数函数: percentile\_approx 64](#_Toc464056346)

[16.8.14 直方图: histogram\_numeric 64](#_Toc464056347)

[16.9 复合类型构建操作 64](#_Toc464056348)

[16.9.1 Map类型构建: map 64](#_Toc464056349)

[16.9.2 Struct类型构建: struct 65](#_Toc464056350)

[16.9.3 array类型构建: array 65](#_Toc464056351)

[16.10 复杂类型访问操作 65](#_Toc464056352)

[16.10.1 array类型访问: A[n] 65](#_Toc464056353)

[16.10.2 map类型访问: M[key] 65](#_Toc464056354)

[16.10.3 struct类型访问: S.x 66](#_Toc464056355)

[16.11 复杂类型长度统计函数 66](#_Toc464056356)

[16.11.1 Map类型长度函数: size(Map<K.V>) 66](#_Toc464056357)

[16.11.2 array类型长度函数: size(Array<T>) 66](#_Toc464056358)

[16.11.3 类型转换函数 67](#_Toc464056359)

[16.12 自定义函数 67](#_Toc464056360)

[17 常用参考资料路径 68](#_Toc464056361)

# 绪论

## 编写背景

由于项目交付中心，将来的会用到大量的Hadoop、Hive的需求，故在此把一些应用经验，供大家分享，学习。

## 文档说明

* 阅读权限

本文档仅限于核心员工交流，在未经相关负责人许可下不得转阅他人。

* 文档解释权

项目交付中心

# Hive简介

## Hive定义

Hive 是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL），这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。Hive 定义了简单的类 SQL 查询语言，称为 HQL，它允许熟悉 SQL 的用户查询数据。同时，这个语言也允许熟悉 MapReduce 开发者的开发自定义的 mapper 和 reducer 来处理内建的 mapper 和 reducer 无法完成的复杂的分析工。

# Hive体系架构

## 用户接口

用户接口主要有三个：CLI，Client 和 WUI。其中最常用的是 CLI，Cli 启动的时候，会同时启动一个 Hive 副本。Client 是 Hive 的客户端，用户连接至 Hive Server。在启动 Client 模式的时候，需要指出 Hive Server 所在节点，并且在该节点启动 Hive Server。 WUI 是通过浏览器访问 Hive。

## 元数据存储

Hive 将元数据存储在数据库中，如 mysql、derby。Hive 中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等。

解释器、编译器、优化器、执行器完成 HQL 查询语句从词法分析、语法分析、编译、优化以及查询计划的生成。生成的查询计划存储在 HDFS 中，并在随后由 MapReduce 调用执行。

## Hadoop

Hive 的数据存储在 HDFS 中，大部分的查询由 MapReduce 完成（不包含 \* 的查询，比如 select \* from tbl 不会生成 MapReduce 任务）。

## Hive 的数据存储

Hive 中所有的数据都存储在 HDFS 中，Hive 没有专门的数据存储格式，也没有为数据建立索引，用户可以非常自由的组织 Hive 中的表，只需要在创建表的时候告诉 Hive 数据中的列分隔符和行分隔符，Hive 就可以解析数据。

# Hive数据类型

## 基本数据类型

|  |  |  |
| --- | --- | --- |
| 数据类型 | 所占字节 | 开始支持版本及描述 |
| TINYINT | 1个字节（8位）有符号整数 | 1 |
| SMALLINT | 2字节（16位）有符号整数 | 1 |
| INT | 4字节（32位）有符号整数 | 1 |
| BIGINT | 8字节（64位）有符号整数 | 1 |
| FLOAT | 4字节（32位）单精度浮点数 | 1 |
| DOUBLE | 8字节（64位）双精度浮点数 | 1 |
| BOOLEAN | true/false | TRUE |
| STRING | 字符串 | ‘xia’,”xia” |
| BINARY |  | 从Hive0.8.0开始支持 |
| TIMESTAMP |  | 从Hive0.8.0开始支持 |
| DECIMAL |  | 从Hive0.11.0开始支持 |
| CHAR |  | 从Hive0.13.0开始支持 |
| VARCHAR |  | 从Hive0.12.0开始支持 |
| DATE |  | 从Hive0.12.0开始支持 |

　　hive是用java开发的，hive里的基本数据类型和java的基本数据类型也是一一对应的，除了string类型。有符号的整数类型：TINYINT、SMALLINT、INT和BIGINT分别等价于java的byte、short、int和long原子类型，它们分别为1字节、2字节、4字节和8字节有符号整数。Hive的浮点数据类型FLOAT和DOUBLE,对应于java的基本类型float和double类型。而hive的BOOLEAN类型相当于java的基本数据类型boolean。

对于hive的String类型相当于数据库的varchar类型，该类型是一个可变的字符串，不过它不能声明其中最多能存储多少个字符，理论上它可以存储2GB的字符数。

Hive支持基本类型的转换，低字节的基本类型可以转化为高字节的类型，例如TINYINT、SMALLINT、INT可以转化为FLOAT，而所有的整数类型、FLOAT以及STRING类型可以转化为DOUBLE类型，这些转化可以从java语言的类型转化考虑，因为hive就是用java编写的。当然也支持高字节类型转化为低字节类型，这就需要使用hive的自定义函数CAST了.

## 复杂数据类型

复杂数据类型包括数组（ARRAY）、映射（MAP）和结构体（STRUCT），具体如下表所示：

|  |  |  |
| --- | --- | --- |
| 复杂数据类型 | | |
| 类型 | 描述 | 示例 |
| ARRAY | 一组有序字段。字段的类型必须相同 | Array(1,2) |
| MAP | 一组无序的键/值对。键的类型必须是原子的，值可以是任何类型，同一个映射的键的类型必须相同，值得类型也必须相同 | Map(‘a’,1,’b’,2) |
| STRUCT | 一组命名的字段。字段类型可以不同 | Struct(‘a’,1,1,0) |

## 数据类型建议

在O域项目中字符串应用String，指标类为BIGINT。

# Hive存储格式

hive文件存储格式包括以下几类：TEXTFILE、SEQUENCEFILE、RCFILE、自定义格式

其中TEXTFILE为默认格式，建表时不指定默认为这个格式，导入数据时会直接把数据文件拷贝到hdfs上不进行处理。SequenceFile,RCFile格式的表不能直接从本地文件导入数据，数据要先导入到textfile格式的表中，然后再从TextFile表中用insert导入到SequenceFile,RCFile表中。

## TEXTFIEL

默认格式，数据不做压缩，磁盘开销大，数据解析开销大。可结合Gzip、Bzip2使用（系统自动检查，执行查询时自动解压），但使用这种方式，hive不会对数据进行切分，从而无法对数据进行并行操作。

示例：

|  |
| --- |
| hive (default) > create table user\_id (user\_id string) |
| > stored as TextFile; |
| OK |
| Time taken: 0.143 seconds |

## SEQUENCEFILE

SequenceFile是Hadoop API提供的一种二进制文件支持，其具有使用方便、可分割、可压缩的特点。SequenceFile支持三种压缩选择：NONE, RECORD, BLOCK。 Record压缩率低，一般建议使用BLOCK压缩。

示例：

|  |
| --- |
| hive (default)> create table user\_id (user\_id string) |
| > stored as SequenceFile; |
| OK |
| Time taken: 0.14 seconds |

## RCFILE

RCFILE是一种行列存储相结合的存储方式。首先，其将数据按行分块，保证同一个record在一个块上，避免读一个记录需要读取多个block。其次，块数据列式存储，有利于数据压缩和快速的列存取。

示例：

|  |
| --- |
| hive (default)> create table user\_id (user\_id string) |
| > stored as RcFile;; |
| OK |
| Time taken: 0.174 seconds |

## 自定义格式

当用户的数据文件格式不能被当前 Hive 所识别的时候，可以自定义文件格式。用户可以通过实现inputformat和 outputformat来自定义输入输出格式，由于目前使用的Hadoop版本为CDH，无法找到源代码，自定义数据格式(如有解决方法，及时更新)

## 存储格式建议

在O域项目中，正式表中应用的存储格式为Rcfile，如是临时需求应用TEXTFILE格式。

# Hive编解码器(压缩格式)

Hadoop版本已经内置支持Gzip和Baip2压缩方案，包括加速对这些个事的压缩和解压的本地Liunx库，Snappy和Lzo不是Hadoop内置的方案，需要自行增加。

使用压缩的优势是可以最小化所需要的磁盘存储空间，以及减小磁盘和网络I/O操作，不过，文件压缩过程和解压缩过程会增加CPU开销。因此，对于密集型压缩的job最好使用压缩，特别是有额外的CPU资源或磁盘存储空间比较稀缺的情况。

每一个压缩方案都在压缩和解压速度和解压率之间进行权衡。

## 压缩格式对比

|  |  |  |  |
| --- | --- | --- | --- |
| 压缩方式 | 压缩比 | 优点 | 缺点 |
| DefaultCodec | 1:1 | 无需解压 | 存储大 |
| BZip2Codec | 1:5 | 压缩比最高 支持块分割 | Cpu开销最多 |
| GzipCodec | 1:3.5 | 压缩比高 | CPU开销多 不支持分割 |
| LzopCodec | 1:2.5 | 解压速度快 支持分割（需建索引） | 压缩比小 |
| SnappyCodec | 1:2.5 | 解压速度快 | 压缩比小 不支持分割 |

基于用户所使用的Hadoop版本，会提供不同的编解码器,查看编解码器的方法两种，

使用Hive的set io.compression.codecs;再者查看Hadoop集群的配置文件

示例：

hive (default)> set io.compression.codecs;

io.compression.codecs=org.apache.hadoop.io.compress.DefaultCodec,

org.apache.hadoop.io.compress.GzipCodec,

org.apache.hadoop.io.compress.BZip2Codec,

com.hadoop.compression.lzo.LzopCodec,

org.apache.hadoop.io.compress.SnappyCodec

## 压缩格式建议

在O域项目中，由于O域的数据量比较大，考虑到集群整体性能的因素，应用的压缩格式为Snappy。其他项目可以根据各自的数据量所选择，Lzo是个不错的选择。

# HiveQL:数据定义

HiveQL是Hive查询语言。和普通使用的所有SQL方言一样，他不上网安全遵守任何一种ANSISQL表针的修订版。HiveQL和MySQL的方言最近进，但是两者还是存在显著性的差异。Hive不支持行级插入操作、更新、和删除操作。Hive也不支持事物。Hive增加了在Hadoop北京下的可以提供跟高性能的扩展，一级一些个性化的扩展，甚至还增加一些外部程序。

## Hive中的数据库

Hive中数据库的概念本质上仅仅是表的一个目录或者命名空间。这样对于应用Hive搭建数据仓库，具有很多组合用户的集群来说，这是非常有用的，因为这样可以避免表命名冲突。通常会使用数据库来生产表组织成逻辑组。

### 创建数据库

示例：

hive (default)> create database ZB\_ODS;

OK

Time taken: 1.013 seconds

如果数据库ZB\_ODS已经存在的话，将会抛出一个错误信息，使用以下语句可以避免在这中情况下抛出错误信息

hive (default)> create database ZB\_ODS;

FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. Database ZB\_ODS already exists

hive (default)> create database if not exists ZB\_ODS;

OK

Time taken: 0.’03’6 seconds

### 查看数据库

示例：

hive (default)> show databases;

OK

default

zb\_dim

zb\_dwa

zb\_dwd

zb\_ods

zb\_src

Time taken: 0.059 seconds, Fetched: 7 row(s)

### 删除数据库

示例：

hive (default)> drop database zb\_ods;

OK

Time taken: 0.372 seconds

默认情况下，Hive是不允许用户删除一个包含有表的数据库的。用户耀目显现出数据库中的表，谈后再删除数据库；要么在删除命令的最后边加上关键字CASCADE,这样可以使Hive自行先删除数据库中的表，再删除数据库

示例：

hive (zb\_ods)> drop database zb\_ods ;

FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. InvalidOperationException(message:Database zb\_ods is not empty)

hive (zb\_ods)> drop database zb\_ods cascade;

OK

Time taken: 0.299 seconds

### 进入数据库

示例：

hive (default)> use zb\_ods;

OK

Time taken: 0.’03’2 seconds

hive (zb\_ods)>

## Hive的表

Hive中的表可分为管理表和外部表。

### 管理表

管理表中的数据存储有配置项hive.metastore.warehouse.dir(默认,/user/hive/warehouse)所定义的目录的子目录下。当删除一个管理表时，Hive也会删除这个表中的数据。

示例：

hive (default)> CREATE TABLE user\_info(

> month\_id string,

> day\_id string,

> user\_id string,

> device\_number string )

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY '\t'

> STORED AS TEXTFILE;

OK

Time taken: 0.158 seconds

### 查看表

示例：

hive (default)> show tables;

OK

user\_info

Time taken: 0.065 seconds, Fetched: 1 row(s)

### 外部表

外部表中的数据存储位置取决于用户创建外部表时指定的LOCATION的位置，创建外部表时需要关键字EXTENAL告诉Hive这个表时外部表，在删除外部表时，并不会删除掉数据，不过描述表的元数据信息会被删除掉。

示例：

hive (default)> CREATE EXTERNAL TABLE USER\_INFO\_EXTENAL(

> month\_id string,

> day\_id string,

> user\_id string,

> device\_number string )

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY '\t'

> STORED AS TEXTFILE

> location '/user/yimr';

OK

Time taken: 0.114 seconds

### 管理分区表

数据分区目的是为了水平分散压力，将数据从物理上转移到和最频繁的用户更近的地方，以及实现其他目的。

Hive中有分区的概念。分区变改变了Hive对数据存储的组织方式，反映在分区结构的子目录，分区字段一旦创建好，与其普通的字段一样，在使用分区时，用WHERE子句中过滤分区字段即可

示例：

hive (default)> CREATE TABLE user\_id(

> device\_number string )

> PARTITIONED BY (

> month\_id string,

> prov\_id string,

> day\_id string);

OK

Time taken: 0.267 seconds

查看分区数据

示例：

hive (default)> select \* from user\_id where month\_id='‘201408’' and prov\_id='010' and day\_id='01' limit 1;

OK

Time taken: 0.726 seconds

这样查看的只有’201408’月份省份为010天为01这天数据

### 外部分区表

外部表同样可以使用分区，事实上，用户可能发现，管理大型生产数据集最常用的情况。这种结合给用户提供了一个可以行和其他工具共享数据的方式，同事也可以优化查询性能。

因为用户可以自己定义目录结构，因此用户对于目录结构的使用具有更多的灵活性。

示例：

hive (default)> CREATE EXTERNAL TABLE user\_info\_extenal(

> month\_id string,

> day\_id string,

> user\_id string,

> device\_number string)

> PARTITIONED BY (

> date\_id string,

> prov\_id string)

> ROW FORMAT DELIMITED

> FIELDS TERMINATED BY '\t' ;

OK

Time taken: 0.497 seconds

给外部分区表中添加数据

hive(default)>alter table user\_info\_extenal add partition(date\_id=‘20140808’,prov\_id=‘011’) location '/user/yimr/’201408’08/’011’';

OK

Time taken: 0.527 seconds

### 删除表

Hive支持和SQL中DROP TABLE 命令类似的操作

示例：

hive (default)> drop table user\_info;

OK

Time taken: 0.398 seconds

可以选择是否使用IF EXITST关键字。如果没有使用这个关键字而且表并不存在的话，那么将会抛出一个错误的信息。

对于管理表，表的元数据信息和表内的数据都会被删除，而外部表只删除元数据信息。

修改表大多数的表属性可以通过ALTER TABLE 语句来进行修改。这种操作会修改元数据，但不会修改数据本身。

### 表重命名

hive (default)> alter table user\_id rename to user\_info;

OK

Time taken: 0.365 seconds

### 增加、修改、和删除表分区

#### 增加分区

hive (default)> alter table user\_info add partition(month\_id=‘201408’,prov\_id=‘011’,day\_id=‘03’);

OK

Time taken: 0.446 seconds

#### 修改分区

hive (default)> alter table user\_info partition(month\_id=‘201408’,prov\_id=‘011’,day\_id=‘03’) rename to partition(month\_id=‘201408’,prov\_id=‘011’,day\_id=04);

OK

Time taken: 0.526 seconds

#### 删除分区

hive(default)>alter table user\_info drop partition(month\_id=‘201408’,prov\_id=‘011’,day\_id=‘03’);

OK

Time taken: 0.444 seconds

#### 修改列信息

alter table user\_info change column home\_number home\_num string after area\_id;

OK

Time taken: 0.347 seconds

#### 增加列

hive (default)> alter table user\_info add columns (area\_id string );

OK

Time taken: 0.283 seconds

#### 删除或替换列

hive (default)> alter table user\_info replace columns ( home\_number string );

OK

Time taken: 0.289 seconds

#### 修改表属性

hive (default)> alter table user\_info set tblproperties ( 'user\_info'='yonghu');

OK

Time taken: 0.276 seconds

#### 查看表属性

hive (default)> show tblproperties user\_info;

OK

user\_info yonghu

last\_modified\_by hadoop

last\_modified\_time 1407463858

transient\_lastDdlTime 1407463858

Time taken: 0.153 seconds, Fetched: 5 row(s)

#### 修改存储属性

hive (default)> alter table user\_info partition ( month\_id=‘201408’,prov\_id=‘011’,day\_id=’02’) set fileformat rcfile;

OK

Time taken: 0.365 seconds

# HiveQL:数据操作

## 向管理表中装载数据

Hive没有行级别的数据插入、数据更新和删除操作，那么往表中装载数据的唯一途径就是使用一种”大量”的数据装在操作。

示例：

如果分区目录不存在，这个命令会先创建分区目录，然后再将数据拷贝到该目录下。如果目标表是非分区表，那么语句中应该省略PARTITION子句。

如果使用了LOCAL这个关键字，那么这个路径应该为本地文件系统路径，数据将会被拷贝到目标位置。如果省略掉LOCAL关键字，那么这个路径应该是分布式文件系统中的路径，这种情况下，数据是从这个路径转移到目标位置。

如果用户指定了OVERWRITE关键字，那么目标文件夹中之前存在的数据将会被先删除掉。如果没有这个关键字，仅仅是把新增的文件增加到文件夹中而不会删除之前的数据。

然而，如果目标文件夹中已经存在和装载的文件同名的文件，那么旧的同名文件将会被股改重写。

Hive并不会验证用户装载的数据和表的模式是否匹配，但Hive会验证文件格式是否和表结构定义的一致。例如，如果表的创建定义的存储格式是RCfile，那么装载进去的文件也应该是RCfile格式的才行，否则抛出异常

## 通过查询语句向表中插入数据

INSERT语句允许用户通过查询语句向目标表中插入数据。

示例:

hive (default)> insert overwrite table user\_id partition(month\_id='201408',prov\_id='010',day\_id='08')

> select home\_number from user\_info ;

这里使用了OVERWRITE关键字，因此之前分区中的内容(如果是非分区表，就是之议案表中的数据)将会被覆盖掉。

这里如果没有使用OVERWRITE关键字或者使用INTO关键字替换掉，那么Hive将会以追加的方式写入数据而不会覆盖掉之前已经存在的内容。这个功能只有Hivev0.8.0版本以及之后的版本中才有。

Hive提供了另一种INSERT语法

示例：

hive (default)> from user\_info

> insert overwrite table user\_id partition(month\_id='201408',prov\_id='010',day\_id='08')

> select home\_number

> insert overwrite table user\_number partition(month\_id='201408',prov\_id='010',day\_id='08')

> select home\_number;

这样只会扫描一次输入数据，然后按多种方式划分，及一个来源数据可以插入多个目标表，也可以为一个表多个分区，这样大大提高了加工效率。

## 导出数据

Hive如何从表中导出数据，在项目中常用的方式有三种

第一种直接从HDFS中导出到本地

示例：

hadoop fs -get /user/dim/dim\_w3\_gprs\_product\_type.txt /bonc/bonc/shell/zb\_test/data

第二种方式从表中导出到本地

示例：

hive (zb\_dim)> insert overwrite local directory '/bonc/hadoop/shell/default'

> row format delimited

> fields terminated by '\t'

> select \* from dim\_w3\_gprs\_product\_type;

Total MapReduce jobs = 1

Launching Job 1 out of 1

Number of reduce tasks is set to 0 since there's no reduce operator

Starting Job = job\_1407211020827\_0036, Tracking URL = http://BD-JK01:23188/proxy/application\_1407211020827\_0036/

Kill Command = /boncemc/boh-2.0.0/hadoop/bin/hadoop job -kill job\_1407211020827\_0036

Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0

2014-08-09 18:07:11,536 Stage-1 map = 0%, reduce = 0%

2014-08-09 18:07:20,645 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 3.74 sec

2014-08-09 18:07:21,758 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 3.74 sec

MapReduce Total cumulative CPU time: 3 seconds 740 msec

Ended Job = job\_1407211020827\_0036

Copying data to local directory /bonc/hadoop/shell/default

Copying data to local directory /bonc/hadoop/shell/default

MapReduce Jobs Launched:

Job 0: Map: 1 Cumulative CPU: 3.74 sec HDFS Read: 2282 HDFS Write: 1908 SUCCESS

Total MapReduce CPU Time Spent: 3 seconds 740 msec

OK

Time taken: 22.257 seconds

需要注意的是，这种方式是需要启用Mapreduce完成，最后生成的文件个数是有Reduce的个数决定的，一般情况下，需把文件合并

第三种直接从表中导出到本地

示例：

hive -e "select \* from zb\_dim.dim\_w3\_gprs\_product;">/bonc/hadoop/shell/default/dim\_w3\_gprs\_product.txt

需要注意的是这种方式数据之间的分隔符为默认的TAB键，其执行过程也是通过Mapreduce完成，但个数只有一个。”>”代表重写，”>>”代表追加。

## 导出数据建议

如果导出数据到本地，则选择第三种，如果数据需要交互到Oracle或者Gp则选择Java编写导数包，来实现数据交换。或者使用总部的DATAX来实现数据交互。

# HiveQL:查询

## 避免Mapruduce查询

hive (zb\_dwd)> select \* from user\_id limit 1;

OK

14510812944

Time taken: 1.608 seconds, Fetched: 1 row(s)

这种情况下，Hive可以简单地读取user\_id对应的存储目录下的文件，然后输出格式化后的内容到控制台。

对于WHERE语句中的过滤条件只是分区字段情况，也是无需MapRuduce过程

hive (zb\_dwd)> select \* from user\_id where date\_id='20140512' limit 1;

OK

14510812944

Time taken: 0.782 seconds, Fetched: 1 row(s)

此外，如果属性hive.exec.mode.local.auto的值设为true,Hive还会尝试使用本地模式执行其他的操作。

hive (zb\_dwd)> set hive.exec.mode.local.auto=true;

hive (zb\_dwd)> select device\_number from user\_id where date\_id='20140512' limit 1;

Automatically selecting local only mode for query

Total MapReduce jobs = 1

Launching Job 1 out of 1

Number of reduce tasks is set to 0 since there's no reduce operator

SLF4J: Class path contains multiple SLF4J bindings.

SLF4J: Found binding in [jar:file:/boncemc/boh-2.0.0/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: Found binding in [jar:file:/boncemc/boh-2.0.0/hive/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]

SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.

SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]

14/08/10 15:56:23 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

14/08/10 15:56:24 WARN conf.Configuration: file:/tmp/hadoop/hive\_2014-08-10\_15-56-19\_828\_8346926598841391064-2/-local-10002/jobconf.xml:an attempt to override final parameter: mapreduce.job.end-notification.max.retry.interval; Ignoring.

14/08/10 15:56:24 WARN conf.Configuration: file:/tmp/hadoop/hive\_2014-08-10\_15-56-19\_828\_8346926598841391064-2/-local-10002/jobconf.xml:an attempt to override final parameter: mapreduce.job.end-notification.max.attempts; Ignoring.

14/08/10 15:56:24 INFO Configuration.deprecation: mapred.input.dir.recursive is deprecated. Instead, use mapreduce.input.fileinputformat.input.dir.recursive

14/08/10 15:56:24 INFO Configuration.deprecation: mapred.max.split.size is deprecated. Instead, use mapreduce.input.fileinputformat.split.maxsize

14/08/10 15:56:24 INFO Configuration.deprecation: mapred.min.split.size is deprecated. Instead, use mapreduce.input.fileinputformat.split.minsize

14/08/10 15:56:24 INFO Configuration.deprecation: mapred.min.split.size.per.rack is deprecated. Instead, use mapreduce.input.fileinputformat.split.minsize.per.rack

14/08/10 15:56:24 INFO Configuration.deprecation: mapred.min.split.size.per.node is deprecated. Instead, use mapreduce.input.fileinputformat.split.minsize.per.node

14/08/10 15:56:24 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduces

14/08/10 15:56:24 INFO Configuration.deprecation: mapred.reduce.tasks.speculative.execution is deprecated. Instead, use mapreduce.reduce.speculative

Execution log at: /tmp/hadoop/hadoop\_20140810155656\_60fe9e2e-4214-4188-a1fd-d1f8e18c21c9.log

Job running in-process (local Hadoop)

2014-08-10 15:56:28,756 null map = 0%, reduce = 0%

2014-08-10 15:56:29,792 null map = 100%, reduce = 0%

Ended Job = job\_local635521341\_0001

Execution completed successfully

MapredLocal task succeeded

OK

14510812944

Time taken: 12.716 seconds, Fetched: 1 row(s)

简单查询不走MapReduce参数set hive.fetch.task.conversion=more;

hive (zb\_dwd)> set hive.fetch.task.conversion=more;

hive (zb\_dwd)> select device\_number from dwd\_d\_wap\_mb\_all\_part where date\_id='20140512' limit 1;

OK

14510812944

Time taken: 0.35 seconds, Fetched: 1 row(s)

## Join语句

Hive支持通常的SQL JOIN语句，但是支持等值关联。

### INNER JOIN

内连接(INNER JOIN)中，只有进行连接的两个表中都存在与连接标准相匹配的数据才会被保留下来。标准SQL是支持对连接关键字进行非等值连接的，但非等值连接在Hive中是非法的，主要原因是通过MapReduce很难实现这种类型的连接，同时，Hive目前还不支持在ON子句中的谓词间使用OR。

大多数情况下，Hive会对每对JOIN连接对象启动一个MapReduce任务。

### LEFT OUTER JOIN

左外连接，这用JOIN连接操作中，JOIN操作符左边表中符合WHERE子句的所有记录将会被返回。JOIN才做符邮编表中如果没有符合ON后边连接条件的记录时，那么从右边表制定选择的列的值将会是NULL

### RIGHT OUTER JOIN

右外连接会返回右边表中所有符合WHERE语句中的记录，左表中匹配不上的字段值用NULL代替。

### FULL OUTER JOIN

完全外连接，将会返回多有表中符合WHERE语句条件的多有记录，如果任一表的制定字段没有符合条件的值的话，那么就使用NULL值代替

### LEFT SEMI JOIN

左半开连接会返回左表中的记录，SENI JOIN比通常的INNER JOIN要更高效，其原因是对于左表一条制定的记录，在右表中一旦找到匹配的记录，Hive就会理解停止扫描。注意的是SELECTH和WHERE语句中不能引用到右边表中的字段。

### MAP SIDE JOIN

如果所有表中只有一张是小表，那么可以在最大的表通过MAPPER的时候讲小表完全放到内存中，Hive可以在map段执行廉洁过程(称为MAP SDIE JOIN)，这是因为Hive可以和内存中的小表进行注意匹配，从而省略掉常规连接操作所需要的RREDUCE

## [Order by, Sort by ,Dristribute by,Cluster By](http://www.cnblogs.com/ggjucheng/archive/2013/01/03/2843243.html)

### order by

order by 会对输入做全局排序，因此只有一个reducer（多个reducer无法保证全局有序）  
只有一个reducer，会导致当输入规模较大时，需要较长的计算时间。

set hive.mapred.mode=nonstrict; (default value / 默认值)

set hive.mapred.mode=strict;

order by 和数据库中的Order by 功能一致，按照某一项 & 几项 排序输出。

与数据库中 order by 的区别在于在hive.mapred.mode = strict 模式下 必须指定 limit 否则执行会报错。

hive> select \* from test order by id;

FAILED: Error in semantic analysis: 1:28 In strict mode, if ORDER BY is specified, LIMIT must also be specified. Error encountered near token 'id'

原因： 在order by 状态下所有数据会到一台服务器进行reduce操作也即只有一个reduce，如果在数据量大的情况下会出现无法输出结果的情况，如果进行 limit n ，那只有 n \* map number 条记录而已。只有一个reduce也可以处理过来。

### sort by

sort by不是全局排序，其在数据进入reducer前完成排序.

因此，如果用sort by进行排序，并且设置mapred.reduce.tasks>1， 则sort by只保证每个reducer的输出有序，不保证全局有序。

sort by 不受 hive.mapred.mode 是否为strict ,nostrict 的影响

sort by 的数据只能保证在同一reduce中的数据可以按指定字段排序。

使用sort by 你可以指定执行的reduce 个数 （set mapred.reduce.tasks=<number>）,对输出的数据再执行归并排序，即可以得到全部结果。

注意：可以用limit子句大大减少数据量。使用limit n后，传输到reduce端（单机）的数据记录数就减少到n\* （map个数）。否则由于数据过大可能出不了结果。

### distribute by

按照指定的字段对数据进行划分到不同的输出reduce / 文件中。

insert overwrite local directory '/home/hadoop/out' select \* from test order by name distribute by length(name);

此方法会根据name的长度划分到不同的reduce中，最终输出到不同的文件中。

length 是内建函数，也可以指定其他的函数或这使用自定义函数。

### Cluster By

cluster by 除了具有 distribute by 的功能外还兼具 sort by 的功能。

但是排序只能是倒序排序，不能指定排序规则为asc 或者desc。

# HiveQL:视图

视图可以循序保存一个查询表并像对待表一样对待这个产讯进行操作。这是一个逻辑结构，因为他不像一个表会存储数据。换句话说，Hive目前暂不支持物化视图。

当一个查询引用一个视图时，这个视图多定义的查询语句将和用户的查询语句组合在一起，然后Hive制定查询计划。从逻辑上讲，可以想象为Hive限制性这个视图，然后使用这个结果进行余下后续的查询。

## 创建View

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

[COMMENT view\_comment]

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

AS SELECT ...

## 删除view

DROP VIEW [IF EXISTS] view\_name

## 修改view

ALTER VIEW view\_name SET TBLPROPERTIES table\_properties

table\_properties:

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

# HiveQL:索引

Hive只有有限的索引功能。Hive中没有普通关系型数据库中键的概念，但是还是可以对一些字段简历索引来加速某些操作。一张表的索引数据存储在另外一张表中。

同时，因为这是一个相对比较新的功能，所以目前还没有提供很多的选择。然而，索引处理没款被设计成为可以制定的Java编码的产检，因此，用户可以根据需要对其进行实现，以满足自身的需求。

当逻辑分区实际上太多太细而无法使用是，简历索引也就是成为分区的另外一个选择。建立索引可以版主裁剪掉一张表的一些数据块，这样能够坚守MapReduce的输入数据量。并非所有的查询都可以通过建立索引获得好处，通过EXPLAIN命令可以查看木某个查询语句是否用到索引。

Hive中的索引和哪些关系型数据库中的一样，需要进行仔细评估才能使用个。维护索引也需要额外的存储空间。同时创建索引也需要消耗资源。用户需要在简历索引为查询带来的好出和因此需要付出的待见之间做出平衡。

hive索引具有以下特点：  
1.索引key冗余存储，提供基于key的数据视图  
2.存储设计以优化查询&检索性能  
3.对于某些查询减少IO，从而提高性能。

## 创建索引：

CREATE INDEX index\_name

ON TABLE base\_table\_name (col\_name, ...)

AS 'index.handler.class.name'

[WITH DEFERRED REBUILD]

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

[IN TABLE index\_table\_name]

[PARTITIONED BY (col\_name, ...)]

[

[ ROW FORMAT ...] STORED AS ...

| STORED BY ...

]

[LOCATION hdfs\_path]

[TBLPROPERTIES (...)]

[COMMENT "index comment"]

注意：  
1.index的partition默认和数据表一致  
2.视图上不能创建index  
3. index可以通过stored as配置存储格式

## 重建索引：

ALTER INDEX index\_name ON table\_name [PARTITION (...)] REBUILD

假如在创建索引是我们使用“ WITH DEFERRED REBUILD”语句，则索引创建是为空，可以通过“Alter index ... REBUILD”在一个partition上或所有partition上构建索引。  
注意：  
1.当hive数据更新时，必须调用该语句更新索引。  
2. index rebuild操作时一个原子操作，因此，当rebuild失败时，先前构建的索引也无法使用

## 删除索引

DROP INDEX index\_name ON table\_name

# Hive元数据

默认情况下，Hive元数据保存在内嵌的 Derby 数据库中，只能允许一个会话连接，只适合简单的测试。为了支持多用户多会话，则需要一个独立的元数据库，我们使用 MySQL 作为元数据库，Hive 内部对 MySQL 提供了很好的支持.

## 数据字典

其中主要涉及到的表如下

|  |  |  |
| --- | --- | --- |
| 表名 | 说明 | 关联键 |
| TBLS | 所有hive表的基本信息(表名，创建时间，所属者等) | TBL\_ID,SD\_ID |
| TABLE\_PARAM | 表级属性，（如是否外部表，表注释，最后修改时间等） | TBL\_ID |
| COLUMNS | Hive表字段信息(字段注释，字段名，字段类型，字段序号) | SD\_ID |
| SDS | 所有hive表、表分区所对应的hdfs数据目录和数据格式 | SD\_ID,SERDE\_ID |
| SERDE\_PARAM | 序列化反序列化信息，如行分隔符、列分隔符、NULL的表示字符等 | SERDE\_ID |
| PARTITIONS | Hive表分区信息（所属表，分区值） | PART\_ID,SD\_ID,TBL\_ID |
| PARTITION\_KEYS | Hive分区表分区键（即分区字段） | TBL\_ID |
| PARTITION\_KEY\_VALS | Hive表分区名(键值) | PART\_ID |

# 数据倾斜

在做Shuffle阶段的优化过程中，遇到了数据倾斜的问题，造成了对一些情况下优化效果不明显。主要是因为在Job完成后的所得到的Counters是整个Job的总和，优化是基于这些Counters得出的平均值，而由于数据倾斜的原因造成map处理数据量的差异过大，使得这些平均值能代表的价值降低。Hive的执行是分阶段的，map处理数据量的差异取决于上一个stage的reduce输出，所以如何将数据均匀的分配到各个reduce中，就是解决数据倾斜的根本所在。规避错误来更好的运行比解决错误更高效。在查看了一些资料后，总结如下。

## 数据倾斜的原因

### 操作

|  |  |  |
| --- | --- | --- |
| 关键词 | 情形 | 后果 |
| Join | 其中一个表较小，  但是key集中 | 分发到某一个或几个Reduce上的数据远高于平均值 |
| 大表与大表，但是分桶的判断字段0值或空值过多 | 这些空值都由一个reduce处理，灰常慢 |
| group by | group by 维度过小，  某值的数量过多 | 处理某值的reduce灰常耗时 |
| Count Distinct | 某特殊值过多 | 处理此特殊值的reduce耗时 |

### 原因

1)、key分布不均匀

2)、业务数据本身的特性

3)、建表时考虑不周

4)、某些SQL语句本身就有数据倾斜

### 表现

任务进度长时间维持在99%（或100%），查看任务监控页面，发现只有少量（1个或几个）reduce子任务未完成。因为其处理的数据量和其他reduce差异过大。

单一reduce的记录数与平均记录数差异过大，通常可能达到3倍甚至更多。 最长时长远大于平均时长。

## 数据倾斜的解决方案

### 参数调节

hive.map.aggr = true

Map 端部分聚合，相当于Combiner

hive.groupby.skewindata=true

数据倾斜的时候进行负载均衡，当项设定为 true，生成的查询计划会有两个 MR Job。第一个 MR Job 中，Map 的输出结果集合会随机分布到 Reduce 中，每个 Reduce 做部分聚合操作，并输出结果，这样处理的结果是相同的 Group By Key 有可能被分发到不同的 Reduce 中，从而达到负载均衡的目的；第二个 MR Job 再根据预处理的数据结果按照 Group By Key 分布到 Reduce 中（这个过程可以保证相同的 Group By Key 被分布到同一个 Reduce 中），最后完成最终的聚合操作。

### SQL语句调节

如何Join：

关于驱动表的取，用join key分布最均匀的表作为驱动表

做好列裁剪和filter操作，以达到两表做join的时候，数据量相对变小的效果。

大小表Join：

使用map join让小的维度表（1000条以下的记录条数） 先进内存。在map端完成reduce.

大表Join大表：

把空值的key变成一个字符串加上随机数，把倾斜的数据分到不同的reduce上，由于null值关联不上，处理后并不影响最终结果。

count distinct大量相同特殊值

count distinct时，将值为空的情况单独处理，如果是计算count distinct，可以不用处理，直接过滤，在最后结果中加1。如果还有其他计算，需要进行group by，可以先将值为空的记录单独处理，再和其他计算结果进行union。

group by维度过小：

采用sum() group by的方式来替换count(distinct)完成计算。

特殊情况特殊处理：

在业务逻辑优化效果的不大情况下，些时候是可以将倾斜的数据单独拿出来处理。最后union回去。

### 空值产生的数据倾斜

场景：如日志中，常会信息丢失的问题，比如日志中的 user\_id，如果取其中的 user\_id 和 用户表中的user\_id 关联，会碰到数据倾斜的问题。

解决方法1： user\_id为空的不参与关联（红色字体为修改后）

|  |
| --- |
| select \* from log a  join users b  on a.user\_id is not null  and a.user\_id = b.user\_idunion allselect \* from log a  where a.user\_id is null; |

解决方法2 ：赋与空值分新的key值

|  |
| --- |
| select \*  from log a  left outer join users b  on case when a.user\_id is null then concat(‘hive’,rand() ) else a.user\_id end = b.user\_id; |

结论：方法2比方法1效率更好，不但io少了，而且作业数也少了。解决方法1中 log读取两次，jobs是2。解决方法2 job数是1 。这个优化适合无效 id (比如 -99 , ’’, null 等) 产生的倾斜问题。把空值的 key 变成一个字符串加上随机数，就能把倾斜的数据分到不同的reduce上 ,解决数据倾斜问题。

## 不同数据类型关联产生数据倾斜

场景：用户表中user\_id字段为int，log表中user\_id字段既有string类型也有int类型。当按照user\_id进行两个表的Join操作时，默认的Hash操作会按int型的id来进行分配，这样会导致所有string类型id的记录都分配到一个Reducer中。

解决方法：把数字类型转换成字符串类型

|  |
| --- |
| select \* from users a  left outer join logs b  on a.usr\_id = cast(b.user\_id as string) |

### 小表不小不大，怎么用 map join 解决倾斜问题

使用 map join 解决小表(记录数少)关联大表的数据倾斜问题，这个方法使用的频率非常高，但如果小表很大，大到map join会出现bug或异常，这时就需要特别的处理。 以下例子:

|  |
| --- |
| select \* from log a  left outer join users b  on a.user\_id = b.user\_id; |

users 表有 600w+ 的记录，把 users 分发到所有的 map 上也是个不小的开销，而且 map join 不支持这么大的小表。如果用普通的 join，又会碰到数据倾斜的问题。

解决方法：

|  |
| --- |
| select /\*+mapjoin(x)\*/\* from log a  left outer join (    select  /\*+mapjoin(c)\*/d.\*      from ( select distinct user\_id from log ) c      join users d      on c.user\_id = d.user\_id    ) x  on a.user\_id = b.user\_id; |

假如，log里user\_id有上百万个，这就又回到原来map join问题。所幸，每日的会员uv不会太多，有交易的会员不会太多，有点击的会员不会太多，有佣金的会员不会太多等等。所以这个方法能解决很多场景下的数据倾斜问题。

## 总结

使map的输出数据更均匀的分布到reduce中去，是我们的最终目标。由于Hash算法的局限性，按key Hash会或多或少的造成数据倾斜。大量经验表明数据倾斜的原因是人为的建表疏忽或业务逻辑可以规避的。在此给出较为通用的步骤：

1、采样log表，哪些user\_id比较倾斜，得到一个结果表tmp1。由于对计算框架来说，所有的数据过来，他都是不知道数据分布情况的，所以采样是并不可少的。

2、数据的分布符合社会学统计规则，贫富不均。倾斜的key不会太多，就像一个社会的富人不多，奇特的人不多一样。所以tmp1记录数会很少。把tmp1和users做map join生成tmp2,把tmp2读到distribute file cache。这是一个map过程。

3、map读入users和log，假如记录来自log,则检查user\_id是否在tmp2里，如果是，输出到本地文件a,否则生成<user\_id,value>的key,value对，假如记录来自member,生成<user\_id,value>的key,value对，进入reduce阶段。

4、最终把a文件，把Stage3 reduce阶段输出的文件合并起写到hdfs。

如果确认业务需要这样倾斜的逻辑，考虑以下的优化方案：

1、对于join，在判断小表不大于1G的情况下，使用map join

2、对于group by或distinct，设定 hive.groupby.skewindata=true

3、尽量使用上述的SQL语句调节进行优化

# Hive参数优化

## 本地模式(小任务)

需要满足以下条件  
1.job的输入数据大小必须小于参数：hive.exec.mode.local.auto.inputbytes.max(默认128MB)  
2.job的map数必须小于参数：hive.exec.mode.local.auto.tasks.max(默认4)  
3.job的reduce数必须为0或者1  
hive.exec.mode.local.auto.inputbytes.max=134217728  
hive.exec.mode.local.auto.tasks.max=4  
hive.exec.mode.local.auto=true  
hive.mapred.local.mem：本地模式启动的JVM内存大小

## 并发执行

hive.exec.parallel=true ，默认为false  
hive.exec.parallel.thread.number=8

## Strict Mode

hive.mapred.mode=true，严格模式不允许执行以下查询：  
分区表上没有指定了分区  
没有limit限制的order by语句  
笛卡尔积：JOIN时没有ON语句

## 动态分区：

hive.exec.dynamic.partition.mode=strict：该模式下必须指定一个静态分区  
hive.exec.max.dynamic.partitions=1000  
hive.exec.max.dynamic.partitions.pernode=100：在每一个mapper/reducer节点允许创建的最大分区数  
DATANODE：dfs.datanode.max.xceivers=8192：允许DATANODE打开多少个文件

## 推测执行：

map red.map.tasks.speculative.execution=true  
mapred.reduce.tasks.speculative.execution=true  
hive.mapred.reduce.tasks.speculative.execution=true;

## Single MapReduce MultiGROUP BY

hive.multigroupby.singlemar=true：当多个GROUP BY语句有相同的分组列，则会优化为一个MR任务

## 是否提供虚拟列

hive.exec.rowoffset

## 分组

两个聚集函数不能有不同的DISTINCT列，以下表达式是错误的：  
INSERT OVERWRITE TABLE pv\_gender\_agg SELECT pv\_users.gender, count(DISTINCT pv\_users.userid), count(DISTINCT pv\_users.ip) FROM pv\_users GROUP BY pv\_users.gender;  
SELECT语句中只能有GROUP BY的列或者聚集函数。

## Map端部分聚合

hive.map.aggr=true;在map中会做部分聚集操作，效率更高但需要更多的内存。

hive.groupby.mapaggr.checkinterval：在Map端进行聚合操作的条目数目。   
hive.groupby.skewindata=true：数据倾斜时负载均衡，当选项设定为true，生成的查询计划会有两个MRJob。第一个MRJob 中，

Map的输出结果集合会随机分布到Reduce中，每个Reduce做部分聚合操作，并输出结果，这样处理的结果是相同的GroupBy Key  
有可能被分发到不同的Reduce中，从而达到负载均衡的目的；第二个MRJob再根据预处理的数据结果按照GroupBy Key分布到  
Reduce中（这个过程可以保证相同的GroupBy Key被分布到同一个Reduce中），最后完成最终的聚合操作。

## Multi- Group-By Inserts

FROM test  
INSERT OVERWRITE TABLE count1  
SELECT count(DISTINCT test.dqcode)  
GROUP BY test.zipcode  
INSERT OVERWRITE TABLE count2  
SELECT count(DISTINCT test.dqcode)  
GROUP BY test.sfcode;

## 排序

ORDER BY colName ASC/DESC  
hive.mapred.mode=strict时需要跟limit子句  
hive.mapred.mode=nonstrict时使用单个reduce完成排序  
SORT BY colName ASC/DESC ：每个reduce内排序  
DISTRIBUTE BY(子查询情况下使用 )：控制特定行应该到哪个reducer，并不保证reduce内数据的顺序  
CLUSTER BY ：当SORT BY 、DISTRIBUTE BY使用相同的列时。

## 合并小文件

hive.merg.mapfiles=true：合并map输出  
hive.merge.mapredfiles=false：合并reduce输出  
hive.merge.size.per.task=256\*1000\*1000：合并文件的大小  
hive.mergejob.maponly=true：如果支持CombineHiveInputFormat则生成只有Map的任务执行merge  
hive.merge.smallfiles.avgsize=16000000：文件的平均大小小于该值时，会启动一个MR任务执行merge。

## Map/reduce数目

减少map数目：

set mapred.max.split.size=512000000;  
set mapred.min.split.size.per.node=512000000;  
set mapred.min.split.size.per.rack=512000000;  
set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;   
增加map数目：  
当input的文件都很大，任务逻辑复杂，map执行非常慢的时候，可以考虑增加Map数，来使得每个map处理的数据量减少，从而提高任务的执行效率。  
假设有这样一个任务：  
　　select data\_desc, count(1), count(distinct id),sum(case when …),sum(case when ...),sum(…) from a group by data\_desc  
如果表a只有一个文件，大小为120M，但包含几千万的记录，如果用1个map去完成这个任务，肯定是比较耗时的，这种情况下，我们要考虑将这一个文件合理的拆分成多个，这样就可以用多个map任务去完成。  
　　set mapred.reduce.tasks=10;  
　　create table a\_1 as select \* from a distribute by rand(123);  
这样会将a表的记录，随机的分散到包含10个文件的a\_1表中，再用a\_1代替上面sql中的a表，则会用10个map任务去完成。每个map任务处理大于12M（几百万记录）的数据，效率肯定会好很多。

reduce数目设置：  
　参数1：hive.exec.reducers.bytes.per.reducer=1G：每个reduce任务处理的数据量  
　参数2：hive.exec.reducers.max=999(0.95\*TaskTracker数)：每个任务最大的reduce数目  
　reducer数=min(参数2,总输入数据量/参数1)  
　set mapred.reduce.tasks：每个任务默认的reduce数目。典型为0.99\*reduce槽数，hive将其设置为-1，自动确定reduce数目。

# 常用优化方向

1、好的模型设计事半功倍。

2、解决数据倾斜问题。

3、减少job数。

4、设置合理的map reduce的task数，能有效提升性能。(比如，10w+级别的计算，用160个reduce，那是相当的浪费，1个足够)。

5、了解数据分布，自己动手解决数据倾斜问题是个不错的选择。

6、数据量较大的情况下，慎用count(distinct)，count(distinct)容易产生倾斜问题。

7、对小文件进行合并，是行至有效的提高调度效率的方法，

8、优化时把握整体，单个作业最优不如整体最优。

# HiveQL:函数

## 关系运算：

### 等值比较: =

语法：A=B

操作类型：所有基本类型

描述: 如果表达式A与表达式B相等，则为TRUE；否则为FALSE

举例：

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

1

### 不等值比较: <>

语法: A <> B

操作类型: 所有基本类型

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

举例：

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

1

### 小于比较: <

语法: A < B

操作类型: 所有基本类型

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

举例：

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

1

### 小于等于比较: <=

语法: A <= B

操作类型: 所有基本类型

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

举例：

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

1

### 大于比较: >

语法: A > B

操作类型: 所有基本类型

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

举例：

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

1

### 大于等于比较: >=

语法: A >= B

操作类型: 所有基本类型

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

举例：

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

1

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

hive> select \* from lxw\_dual;

OK

2011111209 00:00:00 2011111209

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

2011111209 00:00:00 2011111209 false true false

### 空值判断: IS NULL

语法: A IS NULL

操作类型: 所有类型

描述: 如果表达式A的值为NULL，则为TRUE；否则为FALSE

举例：

hive> select 1 from lxw\_dual where null is null;

1

### 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

举例：

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

1

### LIKE比较: LIKE

语法: A LIKE B

操作类型: strings

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

举例：

hive> select 1 from lxw\_dual where 'football' like 'foot%';

1

hive> select 1 from lxw\_dual where 'football' like 'foot\_\_\_\_';

1

注意：否定比较时候用NOT A LIKE B

hive> select 1 from lxw\_dual where NOT 'football' like 'fff%';

1

### JAVA的LIKE操作: RLIKE

语法: A RLIKE B

操作类型: strings

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

举例：

hive> select 1 from lxw\_dual where 'footbar’ rlike '^f.\*r$’;

1

注意：判断一个字符串是否全为数字：

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

1

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

### REGEXP操作: REGEXP

语法: A REGEXP B

操作类型: strings

描述: 功能与RLIKE相同

举例：

hive> select 1 from lxw\_dual where 'footbar' REGEXP '^f.\*r$';

1

## 数学运算：

### 加法操作: +

语法: A + B

操作类型：所有数值类型

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

举例：

hive> select 1 + 9 from lxw\_dual;

10

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

hive> describe lxw\_dual;

\_c0 double

### 减法操作: -

语法: A –B

操作类型：所有数值类型

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

举例：

hive> select 10 –5 from lxw\_dual;

5

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

hive> describe lxw\_dual;

\_c0 double

### 乘法操作: \*

语法: A \* B

操作类型：所有数值类型

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

举例：

hive> select 40 \* 5 from lxw\_dual;

200

### 除法操作: /

语法: A / B

操作类型：所有数值类型

说明：返回A除以B的结果。结果的数值类型为double

举例：

hive> select 40 / 5 from lxw\_dual;

8.0

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

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

结果为4

hive>select ceil(28.0/6.99999999999999) from lxw\_dual limit 1;

结果为5

### 取余操作: %

语法: A % B

操作类型：所有数值类型

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

举例：

hive> select 41 % 5 from lxw\_dual;

1

hive> select 8.4 % 4 from lxw\_dual;

0.40000000000000036

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

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

0.4

### 位与操作: &

语法: A & B

操作类型：所有数值类型

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

举例：

hive> select 4 & 8 from lxw\_dual;

0

hive> select 6 & 4 from lxw\_dual;

4

### 位或操作: |

语法: A | B

操作类型：所有数值类型

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

举例：

hive> select 4 | 8 from lxw\_dual;

12

hive> select 6 | 8from lxw\_dual;

14

### 位异或操作: ^

语法: A ^ B

操作类型：所有数值类型

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

举例：

hive> select 4 ^ 8 from lxw\_dual;

12

hive> select 6 ^ 4 from lxw\_dual;

2

### 位取反操作: ~

语法: ~A

操作类型：所有数值类型

说明：返回A按位取反操作的结果。结果的数值类型等于A的类型。

举例：

hive> select ~6 from lxw\_dual;

-7

hive> select ~4 from lxw\_dual;

-5

## 逻辑运算：

### 逻辑与操作: AND

语法: A AND B

操作类型：boolean

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

举例：

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

1

### 逻辑或操作: OR

语法: A OR B

操作类型：boolean

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

举例：

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

1

### 逻辑非操作: NOT

语法: NOT A

操作类型：boolean

说明：如果A为FALSE，或者A为NULL，则为TRUE；否则为FALSE

举例：

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

1

## 数值计算

### 取整函数: round

语法: round(double a)

返回值: BIGINT

说明: 返回double类型的整数值部分（遵循四舍五入）

举例：

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

3

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

4

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

hive> describe lxw\_dual;

\_c0 bigint

### 指定精度取整函数: round

语法: round(double a, int d)

返回值: DOUBLE

说明: 返回指定精度d的double类型

举例：

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

3.1416

### 向下取整函数: floor

语法: floor(double a)

返回值: BIGINT

说明: 返回等于或者小于该double变量的最大的整数

举例：

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

3

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

25

### 向上取整函数: ceil

语法: ceil(double a)

返回值: BIGINT

说明: 返回等于或者大于该double变量的最小的整数

举例：

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

4

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

46

### 向上取整函数: ceiling

语法: ceiling(double a)

返回值: BIGINT

说明: 与ceil功能相同

举例：

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

4

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

46

### 取随机数函数: rand

语法: rand(),rand(int seed)

返回值: double

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

举例：

hive> select rand() from lxw\_dual;

0.5577432776034763

hive> select rand() from lxw\_dual;

0.6638336467363424

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

0.7220096548596434

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

0.7220096548596434

### 自然指数函数: exp

语法: exp(double a)

返回值: double

说明: 返回自然对数e的a次方

举例：

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

7.38905609893065

自然对数函数: ln

语法: ln(double a)

返回值: double

说明: 返回a的自然对数

举例：

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

2.0

### 以10为底对数函数: log10

语法: log10(double a)

返回值: double

说明: 返回以10为底的a的对数

举例：

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

2.0

### 以2为底对数函数: log2

语法: log2(double a)

返回值: double

说明: 返回以2为底的a的对数

举例：

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

3.0

### 对数函数: log

语法: log(double base, double a)

返回值: double

说明: 返回以base为底的a的对数

举例：

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

4.0

### 幂运算函数: pow

语法: pow(double a, double p)

返回值: double

说明: 返回a的p次幂

举例：

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

16.0

### 幂运算函数: power

语法: power(double a, double p)

返回值: double

说明: 返回a的p次幂,与pow功能相同

举例：

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

16.0

### 开平方函数: sqrt

语法: sqrt(double a)

返回值: double

说明: 返回a的平方根

举例：

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

4.0

### 二进制函数: bin

语法: bin(BIGINT a)

返回值: string

说明: 返回a的二进制代码表示

举例：

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

111

### 十六进制函数: hex

语法: hex(BIGINT a)

返回值: string

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

举例：

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

11

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

616263

### 反转十六进制函数: unhex

语法: unhex(string a)

返回值: string

说明: 返回该十六进制字符串所代码的字符串

举例：

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

abc

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

-

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

abc

### 进制转换函数: conv

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

返回值: string

说明: 将数值num从from\_base进制转化到to\_base进制

举例：

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

11

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

10001

### 绝对值函数: abs

语法: abs(double a) abs(int a)

返回值: double int

说明: 返回数值a的绝对值

举例：

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

3.9

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

10.9

### 正取余函数: pmod

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

返回值: int double

说明: 返回正的a除以b的余数

举例：

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

1

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

3

### 正弦函数: sin

语法: sin(double a)

返回值: double

说明: 返回a的正弦值

举例：

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

0.7173560908995228

### 反正弦函数: asin

语法: asin(double a)

返回值: double

说明: 返回a的反正弦值

举例：

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

0.8

### 余弦函数: cos

语法: cos(double a)

返回值: double

说明: 返回a的余弦值

举例：

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

0.6216099682706644

### 反余弦函数: acos

语法: acos(double a)

返回值: double

说明: 返回a的反余弦值

举例：

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

0.9

### positive函数: positive

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

返回值: int double

说明: 返回a

举例：

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

-10

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

12

### negative函数: negative

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

返回值: int double

说明: 返回-a

举例：

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

5

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

-8

## 日期函数

### UNIX时间戳转日期函数: from\_unixtime

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

返回值: string

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

举例：

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

20111208

### 获取当前UNIX时间戳函数: unix\_timestamp

语法: unix\_timestamp()

返回值: bigint

说明: 获得当前时区的UNIX时间戳

举例：

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

1323309615

### 日期转UNIX时间戳函数: unix\_timestamp

语法: unix\_timestamp(string date)

返回值: bigint

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

举例：

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

1323234063

### 指定格式日期转UNIX时间戳函数: unix\_timestamp

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

返回值: bigint

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

举例：

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

1323234063

### 日期时间转日期函数: to\_date

语法: to\_date(string timestamp)

返回值: string

说明: 返回日期时间字段中的日期部分。

举例：

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

2011-12-08

### 日期转年函数: year

语法: year(string date)

返回值: int

说明: 返回日期中的年。

举例：

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

2011

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

2012

### 日期转月函数: month

语法: month (string date)

返回值: int

说明: 返回日期中的月份。

举例：

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

12

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

8

### 日期转天函数: day

语法: day (string date)

返回值: int

说明: 返回日期中的天。

举例：

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

8

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

24

### 日期转小时函数: hour

语法: hour (string date)

返回值: int

说明: 返回日期中的小时。

举例：

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

10

### 日期转分钟函数: minute

语法: minute (string date)

返回值: int

说明: 返回日期中的分钟。

举例：

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

3

### 日期转秒函数: second

语法: second (string date)

返回值: int

说明: 返回日期中的秒。

举例：

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

1

### 日期转周函数: weekofyear

语法: weekofyear (string date)

返回值: int

说明: 返回日期在当前的周数。

举例：

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

49

### 日期比较函数: datediff

语法: datediff(string enddate, string startdate)

返回值: int

说明: 返回结束日期减去开始日期的天数。

举例：

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

213

### 日期增加函数: date\_add

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

返回值: string

说明: 返回开始日期startdate增加days天后的日期。

举例：

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

2012-12-18

### 日期减少函数: date\_sub

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

返回值: string

说明: 返回开始日期startdate减少days天后的日期。

举例：

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

2012-11-28

## 条件函数

### If函数: if

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

返回值: T

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

举例：

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

200

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

100

### 非空查找函数: COALESCE

语法: COALESCE(T v1, T v2, …)

返回值: T

说明: 返回参数中的第一个非空值；如果所有值都为NULL，那么返回NULL

举例：

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

100

### 条件判断函数：CASE

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

返回值: T

说明：如果a等于b，那么返回c；如果a等于d，那么返回e；否则返回f

举例：

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

mary

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

tim

### 条件判断函数：CASE

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

返回值: T

说明：如果a为TRUE,则返回b；如果c为TRUE，则返回d；否则返回e

举例：

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

mary

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

tom

## 字符串函数

### 字符串长度函数：length

语法: length(string A)

返回值: int

说明：返回字符串A的长度

举例：

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

7

### 字符串反转函数：reverse

语法: reverse(string A)

返回值: string

说明：返回字符串A的反转结果

举例：

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

gfdecba

### 字符串连接函数：concat

语法: concat(string A, string B…)

返回值: string

说明：返回输入字符串连接后的结果，支持任意个输入字符串

举例：

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

abcdefgh

### 带分隔符字符串连接函数：concat\_ws

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

返回值: string

说明：返回输入字符串连接后的结果，SEP表示各个字符串间的分隔符

举例：

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

abc,def,gh

### 字符串截取函数：substr,substring

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

返回值: string

说明：返回字符串A从start位置到结尾的字符串

举例：

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

cde

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

cde

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

e

### 字符串截取函数：substr,substring

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

举例：

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

cd

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

cd

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

de

### 字符串转大写函数：upper,ucase

语法: upper(string A) ucase(string A)

返回值: string

说明：返回字符串A的大写格式

举例：

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

ABSED

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

ABSED

### 字符串转小写函数：lower,lcase

语法: lower(string A) lcase(string A)

返回值: string

说明：返回字符串A的小写格式

举例：

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

absed

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

absed

### 去空格函数：trim

语法: trim(string A)

返回值: string

说明：去除字符串两边的空格

举例：

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

abc

### 左边去空格函数：ltrim

语法: ltrim(string A)

返回值: string

说明：去除字符串左边的空格

举例：

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

abc

### 右边去空格函数：rtrim

语法: rtrim(string A)

返回值: string

说明：去除字符串右边的空格

举例：

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

abc

### 正则表达式替换函数：regexp\_replace

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

返回值: string

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

举例：

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

fb

### 正则表达式解析函数：regexp\_extract

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

返回值: string

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

举例：

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

the

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

bar

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

foothebar

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

select data\_field,

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

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

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

from pt\_nginx\_loginlog\_st

where pt = '2012-03-26' limit 2;

### URL解析函数：parse\_url

语法: parse\_url(string urlString, string partToExtract [, 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.

举例：

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

facebook.com

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

v1

### json解析函数：get\_json\_object

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

返回值: string

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

举例：

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

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

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

> },

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

> "owner":"amy"

> }

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

amy

### 空格字符串函数：space

语法: space(int n)

返回值: string

说明：返回长度为n的字符串

举例：

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

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

10

### 重复字符串函数：repeat

语法: repeat(string str, int n)

返回值: string

说明：返回重复n次后的str字符串

举例：

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

abcabcabcabcabc

### 首字符ascii函数：ascii

语法: ascii(stringstr)

返回值: int

说明：返回字符串str第一个字符的ascii码

举例：

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

97

### 左补足函数：lpad

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

返回值: string

说明：将str进行用pad进行左补足到len位

举例：

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

tdtdtdtabc

注意：与GP，ORACLE不同，pad 不能默认

### 右补足函数：rpad

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

返回值: string

说明：将str进行用pad进行右补足到len位

举例：

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

abctdtdtdt

### 分割字符串函数: split

语法: split(string str, 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

举例：

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

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

### 集合查找函数: find\_in\_set

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

返回值: int

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

举例：

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

2

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

0

## 合统计函数

### 个数统计函数: count

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

返回值: int

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

举例：

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

20

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

10

### 总和统计函数: sum

语法: sum(col), sum(DISTINCT col)

返回值: double

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

举例：

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

100

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

70

### 平均值统计函数: avg

语法: avg(col), avg(DISTINCT col)

返回值: double

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

举例：

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

50

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

30

### 最小值统计函数: min

语法: min(col)

返回值: double

说明: 统计结果集中col字段的最小值

举例：

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

20

### 最大值统计函数: max

语法: maxcol)

返回值: double

说明: 统计结果集中col字段的最大值

举例：

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

120

### 非空集合总体变量函数: var\_pop

语法: var\_pop(col)

返回值: double

说明: 统计结果集中col非空集合的总体变量（忽略null）

举例：

### 非空集合样本变量函数: var\_samp

语法: var\_samp (col)

返回值: double

说明: 统计结果集中col非空集合的样本变量（忽略null）

举例：

### 总体标准偏离函数: stddev\_pop

语法: stddev\_pop(col)

返回值: double

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

举例：

### 样本标准偏离函数: stddev\_samp

语法: stddev\_samp (col)

返回值: double

说明: 该函数计算样本标准偏离

举例：

### 中位数函数: percentile

语法: percentile(BIGINT col, p)

返回值: double

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

举例：

### 中位数函数: percentile

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

返回值: array<double>

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

举例：

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

### 近似中位数函数: percentile\_approx

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

返回值: double

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

举例：

### 近似中位数函数: percentile\_approx

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

返回值: array<double>

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

举例：

### 直方图: histogram\_numeric

语法: histogram\_numeric(col, b)

返回值: array<struct {‘x’,‘y’}>

说明: 以b为基准计算col的直方图信息。

举例：

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

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

## 复合类型构建操作

### Map类型构建: map

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

说明：根据输入的key和value对构建map类型

举例：

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

hive> describe lxw\_test;

t map<string,string>

hive> select t from lxw\_test;

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

### Struct类型构建: struct

语法: struct(val1, val2, val3, …)

说明：根据输入的参数构建结构体struct类型

举例：

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

hive> describe lxw\_test;

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

hive> select t from lxw\_test;

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

### array类型构建: array

语法: array(val1, val2, …)

说明：根据输入的参数构建数组array类型

举例：

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

hive>describe lxw\_test;

t array<string>

hive> select t from lxw\_test;

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

## 复杂类型访问操作

### array类型访问: A[n]

语法: A[n]

操作类型: A为array类型，n为int类型

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

举例：

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

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

tom mary tim

### map类型访问: M[key]

语法: M[key]

操作类型: M为map类型，key为map中的key值

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

举例：

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

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

mary tom

### struct类型访问: S.x

语法: S.x

操作类型: S为struct类型

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

举例：

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

hive> describe lxw\_test;

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

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

tom tim

## 复杂类型长度统计函数

### Map类型长度函数: size(Map<K.V>)

语法: size(Map<K.V>)

返回值: int

说明: 返回map类型的长度

举例：

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

2

### array类型长度函数: size(Array<T>)

语法: size(Array<T>)

返回值: int

说明: 返回array类型的长度

举例：

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

4

### 类型转换函数

类型转换函数: cast

语法: cast(expr as <type>)

返回值: Expected "=" to follow "type"

说明: 返回array类型的长度

举例：

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

1

## 自定义函数

由于关系型数据库中的函数在Hive中不能完全支撑,所以当项目需要关系型制定的函数时,我们可以通过编写Java来编写自定义函数.编写一个

示例:

package com.example.hive.udf;

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

import org.apache.hadoop.io.Text;

public final class Lower extends UDF {

public Text evaluate(final Text s) {

if (s == null) { return null; }

return new Text(s.toString().toLowerCase());

}

}

在hive中添加包

hive> add jar /home/hjl/sunwg/Lower.jar;

Added /home/hjl/sunwg/Lower.jar to class path

在hive中创建函数

hive> create temporary function lower as ‘com.example.hive.udf.Lower’;

OK

Time taken: 0.407 seconds

使用函数

hive> select lower(‘DBA’) from dual;

>dba

附件为常用Oracle中常用的函数，用Java已经实现。



# 常用参考资料路径

Hive地址

<http://wiki.apache.org/hadoop/Hive>

http://hive.apache.org/

Velocity地址

<http://velocity.apache.org/engine/releases/velocity-1.7/user-guide.html>

Hadoop地址

<http://hadoop.apache.org/>

<http://www.cloudera.com/>

Hadoop中文文档地址

<http://hadoop.apache.org/common/docs/r0.18.2/cn/commands_manual.html>