# 第1章 Hive基本概念

## 1.1 什么是Hive

**1） hive简介**

Hive：由Facebook开源用于解决海量结构化日志的数据统计工具。

Hive是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张表，并提供类SQL查询功能。

**2） Hive本质**：将HQL转化成MapReduce程序

（1）Hive处理的数据存储在HDFS

（2）Hive分析数据底层的实现是MapReduce

（3）执行程序运行在Yarn上

## Hive的优缺点

### 1.2.1 优点

（1）操作接口采用类SQL语法，提供快速开发的能力（简单、容易上手）。

（2）避免了去写MapReduce，减少开发人员的学习成本。

（3）Hive优势在于处理大数据，支持海量数据的分析与计算。

（4）Hive支持用户自定义函数，用户可以根据自己的需求来实现自己的函数。

### 1.2.2 缺点

**1）Hive的HQL表达能力有限**

（1）Hive自动生成的MapReduce作业，通常情况下不够智能化

（2）数据挖掘方面不擅长，由于MapReduce数据处理流程的限制，效率更高的算法却无法实现。

**2）Hive的效率比较低**

（1）Hive的执行延迟比较高，因此Hive常用于数据分析，对实时性要求不高的场合。

（2）Hive调优比较困难，粒度较粗

**3）Hive不支持实时查询和行级别更新**

（1）hive分析的数据是存储在hdfs上，hdfs不支持随机写，只支持追加写，所以在hive中不能delete和update，只能select和insert

## 1.3 Hive架构原理



**1）用户接口：Client**

CLI（command-line interface）、JDBC/ODBC(jdbc访问hive)、WEBUI（浏览器访问hive）

**2）元数据：Metastore**

元数据包括：表名、表所属的数据库（默认是default）、表的拥有者、列/分区字段、表的类型（是否是外部表）、表的数据所在目录等；

默认存储在自带的derby数据库中，推荐使用MySQL存储Metastore

**3）Hadoop**

使用HDFS进行存储，使用MapReduce进行计算。

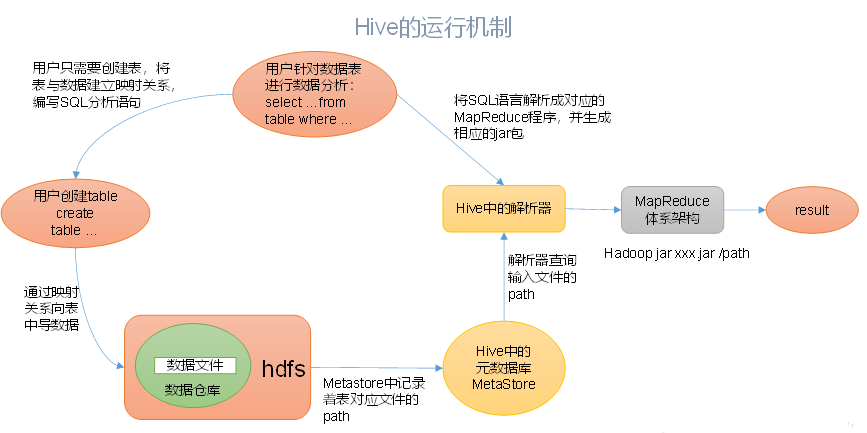
**4）驱动器：Driver**

（1）解析器（SQL Parser）：将SQL字符串转换成抽象语法树AST，这一步一般都用第三方工具库完成，比如antlr；对AST进行语法分析，比如表是否存在、字段是否存在、SQL语义是否有误。

（2）编译器（Physical Plan）：将AST编译生成逻辑执行计划。

（3）优化器（Query Optimizer）：对逻辑执行计划进行优化。

（4）执行器（Execution）：把逻辑执行计划转换成可以运行的物理计划。对于Hive来说，就是MR/Spark。



Hive通过给用户提供的一系列交互接口，接收到用户的指令(SQL)，使用自己的Driver，结合元数据(MetaStore)，将这些指令翻译成MapReduce，提交到Hadoop中执行，最后，将执行返回的结果输出到用户交互接口。

# 第2章 Hive安装

## 2.1 Hive安装地址

**1）Hive官网地址**

http://hive.apache.org/

## Hive安装部署

**1）把apache-hive-3.1.2-bin.tar.gz上传到linux的/opt/software目录下**

**2）解压apache-hive-3.1.2-bin.tar.gz到/opt/module/目录下面**

[user1@hadoop102 software]$ tar -zxvf /opt/software/apache-hive-3.1.2-bin.tar.gz -C /opt/module/

**3）修改apache-hive-3.1.2-bin.tar.gz的名称为hive**

[user1@hadoop102 software]$ mv /opt/module/apache-hive-3.1.2-bin/ /opt/module/hive

**4）修改/etc/profile.d/my\_env.sh，添加环境变量**

[user1@hadoop102 software]$ sudo vim /etc/profile.d/my\_env.sh

**5）添加内容**

#HIVE\_HOME

export HIVE\_HOME=/opt/module/hive

export PATH=$PATH:$HIVE\_HOME/bin

**6）解决日志Jar包冲突**

[user1@hadoop102 software]$ mv $HIVE\_HOME/lib/log4j-slf4j-impl-2.10.0.jar $HIVE\_HOME/lib/log4j-slf4j-impl-2.10.0.bak

## 2.4 Hive元数据配置到MySql

### 2.3.1 拷贝驱动

将MySQL的JDBC驱动拷贝到Hive的lib目录下

[user1@hadoop102 software]$ cp /opt/software/mysql-connector-java-5.1.37.jar $HIVE\_HOME/lib

### 2.3.2 配置Metastore到MySql

在$HIVE\_HOME/conf目录下新建hive-site.xml文件

[user1@hadoop102 software]$ vim $HIVE\_HOME/conf/hive-site.xml

添加如下内容

<?xml version="1.0"?>

<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>

<!-- jdbc连接的URL -->

<property>

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

<value>jdbc:mysql://hadoop102:3306/metastore?useSSL=false</value>

</property>

<!-- jdbc连接的Driver-->

<property>

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

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

</property>

<!-- jdbc连接的username-->

<property>

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

<value>root</value>

</property>

<!-- jdbc连接的password -->

<property>

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

<value>123456</value>

</property>

<!-- Hive默认在HDFS的工作目录 -->

<property>

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

<value>/user/hive/warehouse</value>

</property>

<!-- 指定hiveserver2连接的端口号 -->

<property>

<name>hive.server2.thrift.port</name>

<value>10000</value>

</property>

<!-- 指定hiveserver2连接的host -->

<property>

<name>hive.server2.thrift.bind.host</name>

<value>hadoop102</value>

</property>

<!-- 指定存储元数据要连接的地址 -->

<property>

<name>hive.metastore.uris</name>

<value>thrift://hadoop102:9083</value>

</property>

<!-- 元数据存储授权 -->

<property>

<name>hive.metastore.event.db.notification.api.auth</name>

<value>false</value>

</property>

<!-- Hive元数据存储版本的验证 -->

<property>

<name>hive.metastore.schema.verification</name>

<value>false</value>

</property>

<!-- hiveserver2的高可用参数，开启此参数可以提高hiveserver2的启动速度 -->

<property>

<name>hive.server2.active.passive.ha.enable</name>

<value>true</value>

</property>

</configuration>

## 2.5 启动Hive

### 2.5.1 初始化元数据库

**1）登陆MySQL**

[user1@hadoop102 software]$ mysql -uroot -p123456

**2）新建Hive元数据库**

mysql> create database metastore;

mysql> quit;

**3）初始化Hive元数据库**

[user1@hadoop102 software]$ schematool -initSchema -dbType mysql -verbose

### 2.5.2 启动metastore和hiveserver2

**1）Hive 2.x以上版本，要先启动这两个服务，否则会报错：**

FAILED: HiveException java.lang.RuntimeException: Unable to instantiate org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClient

1. 启动metastore

[user1@hadoop202 hive]$ hive --service metastore

2020-04-24 16:58:08: Starting Hive Metastore Server

注意: 启动后窗口不能再操作，需打开一个新的shell窗口做别的操作

1. 启动 hiveserver2

[user1@hadoop202 hive]$ hive --service hiveserver2

which: no hbase in (/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/module/jdk1.8.0\_212/bin:/opt/module/hadoop-3.1.3/bin:/opt/module/hadoop-3.1.3/sbin:/opt/module/hive/bin:/home/user1/.local/bin:/home/user1/bin)

2020-04-24 17:00:19: Starting HiveServer2

注意: 启动后窗口不能再操作，需打开一个新的shell窗口做别的操作

**2）编写hive服务启动脚本**

1. 前台启动的方式导致需要打开多个shell窗口，可以使用如下方式后台方式启动

nohup: 放在命令开头，表示不挂起,也就是关闭终端进程也继续保持运行状态

2>&1 : 表示将错误重定向到标准输出上

&: 放在命令结尾,表示后台运行

一般会组合使用: nohup [xxx命令操作]> file 2>&1 & ， 表示将xxx命令运行的

结果输出到file中，并保持命令启动的进程在后台运行。

[user1@hadoop102 hive]$ nohup hive --service metastore>log.txt 2>&1 &

[user1@hadoop102 hive]$ nohup hive --service hiveserver2>log2.txt 2>&1 &

1. 为了方便使用，可以直接编写脚本来管理服务的启动和关闭

[user1@hadoop102 hive]$ vim $HIVE\_HOME/bin/hiveservices.sh

#!/bin/bash

HIVE\_LOG\_DIR=$HIVE\_HOME/logs

if [ ! -d $HIVE\_LOG\_DIR ]

then

mkdir -p $HIVE\_LOG\_DIR

fi

#检查进程是否运行正常，参数1为进程名，参数2为进程端口

function check\_process()

{

pid=$(ps -ef 2>/dev/null | grep -v grep | grep -i $1 | awk '{print $2}')

ppid=$(netstat -nltp 2>/dev/null | grep $2 | awk '{print $7}' | cut -d '/' -f 1)

echo $pid

[[ "$pid" =~ "$ppid" ]] && [ "$ppid" ] && return 0 || return 1

}

function hive\_start()

{

metapid=$(check\_process HiveMetastore 9083)

cmd="nohup hive --service metastore >$HIVE\_LOG\_DIR/metastore.log 2>&1 &"

cmd=$cmd" sleep 4; hdfs dfsadmin -safemode wait >/dev/null 2>&1"

[ -z "$metapid" ] && eval $cmd || echo "Metastroe服务已启动"

server2pid=$(check\_process HiveServer2 10000)

cmd="nohup hive --service hiveserver2 >$HIVE\_LOG\_DIR/hiveServer2.log 2>&1 &"

[ -z "$server2pid" ] && eval $cmd || echo "HiveServer2服务已启动"

}

function hive\_stop()

{

metapid=$(check\_process HiveMetastore 9083)

[ "$metapid" ] && kill $metapid || echo "Metastore服务未启动"

server2pid=$(check\_process HiveServer2 10000)

[ "$server2pid" ] && kill $server2pid || echo "HiveServer2服务未启动"

}

case $1 in

"start")

hive\_start

;;

"stop")

hive\_stop

;;

"restart")

hive\_stop

sleep 2

hive\_start

;;

"status")

check\_process HiveMetastore 9083 >/dev/null && echo "Metastore服务运行正常" || echo "Metastore服务运行异常"

check\_process HiveServer2 10000 >/dev/null && echo "HiveServer2服务运行正常" || echo "HiveServer2服务运行异常"

;;

\*)

echo Invalid Args!

echo 'Usage: '$(basename $0)' start|stop|restart|status'

;;

esac

**3）添加执行权限**

[user1@hadoop102 hive]$ chmod +x $HIVE\_HOME/bin/hiveservices.sh

**4）启动Hive后台服务(需要先启动hadoop)**

[user1@hadoop102 hive]$ hiveservices.sh start

### 2.5.3 HiveJDBC访问

**1）启动beeline客户端**

[user1@hadoop102 hive]$ bin/beeline -u jdbc:hive2://hadoop102:10000 -n user1

**2）看到如下界面**

Connecting to jdbc:hive2://hadoop102:10000

Connected to: Apache Hive (version 3.1.2)

Driver: Hive JDBC (version 3.1.2)

Transaction isolation: TRANSACTION\_REPEATABLE\_READ

Beeline version 3.1.2 by Apache Hive

0: jdbc:hive2://hadoop102:10000>

### 2.5.4 Hive访问

**1）启动hive客户端**

[user1@hadoop102 hive]$ bin/hive

**2）看到如下界面**

which: no hbase in (/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/opt/module/jdk1.8.0\_212/bin:/opt/module/hadoop-3.1.3/bin:/opt/module/hadoop-3.1.3/sbin:/opt/module/hive/bin:/home/user1/.local/bin:/home/user1/bin)

Hive Session ID = 36f90830-2d91-469d-8823-9ee62b6d0c26

Logging initialized using configuration in jar:file:/opt/module/hive/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true

Hive Session ID = 14f96e4e-7009-4926-bb62-035be9178b02

hive>

**3）打印 当前库 和 表头**

在hive-site.xml中加入如下两个配置:

<property>

<name>hive.cli.print.header</name>

<value>true</value>

<description>Whether to print the names of the columns in query output.</description>

</property>

<property>

<name>hive.cli.print.current.db</name>

<value>true</value>

<description>Whether to include the current database in the Hive prompt.</description>

</property>

## 2.6 Hive常用交互命令

[user1@hadoop102 hive]$ bin/hive -help

usage: hive

-d,--define <key=value> Variable subsitution to apply to hive

commands. e.g. -d A=B or --define A=B

--database <databasename> Specify the database to use

-e <quoted-query-string> SQL from command line

-f <filename> SQL from files

-H,--help Print help information

--hiveconf <property=value> Use value for given property

--hivevar <key=value> Variable subsitution to apply to hive

commands. e.g. --hivevar A=B

-i <filename> Initialization SQL file

-S,--silent Silent mode in interactive shell

-v,--verbose Verbose mode (echo executed SQL to the console)

**0）在hive命令行里创建一个表student，并插入1条数据**

hive (default)> create table student(id int,name string);

OK

Time taken: 1.291 seconds

hive (default)> insert into student values(1,"zhangsan");

hive (default)> select \* from student;

OK

student.id student.name

1 zhangsan

Time taken: 0.144 seconds, Fetched: 1 row(s)

**1）“-e”不进入hive的交互窗口执行sql语句**

[user1@hadoop102 hive]$ bin/hive -e "select id from student;"

**2）“-f”执行脚本中sql语句**

（1）在/opt/module/hive/下创建datas目录并在datas目录下创建hivef.sql文件

[user1@hadoop102 datas]$ vim hivef.sql

（2）文件中写入正确的sql语句

select \* from student;

（3）执行文件中的sql语句

[user1@hadoop102 hive]$ bin/hive -f /opt/module/hive/datas/hivef.sql

（4）执行文件中的sql语句并将结果写入文件中

[user1@hadoop102 hive]$ bin/hive -f /opt/module/hive/datas/hivef.sql > /opt/module/datas/hive\_result.txt

## 2.7 Hive其他命令操作

**1）退出hive窗口：**

hive(default)>exit;

hive(default)>quit;

在新版的hive中没区别了，在以前的版本是有的：

exit:先隐性提交数据，再退出；

quit:不提交数据，退出；

**2）在hive cli命令窗口中如何查看hdfs文件系统**

hive(default)>dfs -ls /;

**3）查看在hive中输入的所有历史命令**

（1）进入到当前用户的家目录/root或/home/user1

（2）查看. hivehistory文件，注意quit以后，此文件才会写入quit之前提交的命令。

[user1@hadoop102 ~]$ cat .hivehistory

**4）查看在beeline中输入的所有历史命令**

（1）进入到当前用户的家目录/root或/home/user1

（2）查看.beeline/history文件，注意quit以后，此文件才会写入quit之前提交的命令。

[user1@hadoop102 ~]$ cat .beeline/history

## 2.8 Hive常见属性配置

### 2.8.1 Hive运行日志信息配置

**1）Hive的log默认存放在/tmp/user1/hive.log目录下（当前用户名下）**

**2）修改hive的log存放日志到/opt/module/hive/logs**

（1）修改$HIVE\_HOME/conf/hive-log4j.properties.template文件名称为

hive-log4j.properties

[user1@hadoop102 conf]$ pwd

/opt/module/hive/conf

[user1@hadoop102 conf]$ mv hive-log4j.properties.template hive-log4j.properties

（2）在hive-log4j.properties文件中修改log存放位置

hive.log.dir=/opt/module/hive/logs

### 2.8.2 Hive启动jvm堆内存设置

新版本的hive启动的时候，默认申请的jvm堆内存大小为256M，jvm堆内存申请的太小，导致后期开启本地模式，执行复杂的sql时经常会报错：java.lang.OutOfMemoryError: Java heap space，因此最好提前调整一下HADOOP\_HEAPSIZE这个参数。

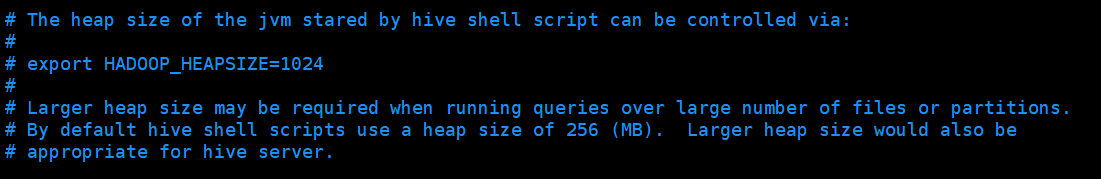
1. 修改$HIVE\_HOME/conf下的hive-env.sh.template为hive-env.sh

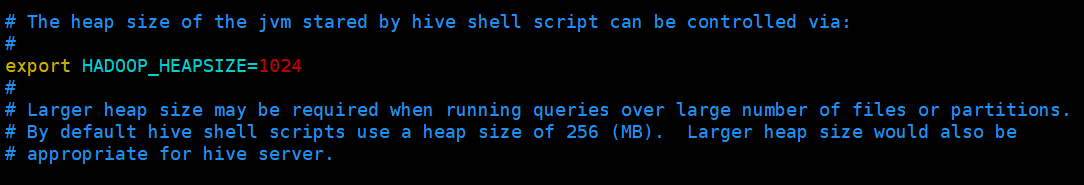
[user1@hadoop102 conf]$ pwd

/opt/module/hive/conf

[user1@hadoop102 conf]$ cp hive-env.sh.template hive-env.sh

（2） 将hive-env.sh其中的参数 export HADOOP\_HEAPSIZE=1024的注释放开，重启hive。





### 2.8.3 参数配置方式

**1）查看当前所有的配置信息**

hive>set;

**2）参数的配置三种方式**

（1）配置文件方式

默认配置文件：hive-default.xml

用户自定义配置文件：hive-site.xml

注意：用户自定义配置会覆盖默认配置。另外，Hive也会读入Hadoop的配置，因为Hive是作为Hadoop的客户端启动的，Hive的配置会覆盖Hadoop的配置。配置文件的设定对本机启动的所有Hive进程都有效。

（2）命令行参数方式

启动Hive时，可以在命令行添加-hiveconf param=value来设定参数。

例如：

[user1@hadoop103 hive]$ bin/hive -hiveconf mapred.reduce.tasks=10;

或者

[user1@hadoop103 hive]$ bin/beeline -u jdbc:hive2://hadoop102:10000 -n user1 -hiveconf mapred.reduce.tasks=10;

注意：仅对本次hive启动有效

查看参数设置：

hive (default)> set mapred.reduce.tasks;

（3）参数声明方式

可以在HQL中使用SET关键字设定参数

例如：

hive (default)> set mapred.reduce.tasks=100;

注意：仅对本次hive启动有效。

查看参数设置

hive (default)> set mapred.reduce.tasks;

上述三种设定方式的优先级依次递增。即配置文件<命令行参数<参数声明。注意某些系统级的参数，例如log4j相关的设定，必须用前两种方式设定，因为那些参数的读取在会话建立以前已经完成了。

# 第3章 Hive数据类型

## 3.1 基本数据类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HIVE | MySQL | JAVA | 长度 | 例子 |
| TINYINT | TINYINT | byte | 1byte有符号整数 | 2 |
| SMALINT | SMALINT | short | 2byte有符号整数 | 20 |
| INT | INT | int | 4byte有符号整数 | 20 |
| BIGINT | BIGINT | long | 8byte有符号整数 | 20 |
| BOOLEAN | 无 | boolean | 布尔类型，true或者false | TRUE FALSE |
| FLOAT | FLOAT | float | 单精度浮点数 | 3.14159 |
| DOUBLE | DOUBLE | double | 双精度浮点数 | 3.14159 |
| STRING | VARCHAR | string | 字符系列。可以指定字符集。可以使用单引号或者双引号。 | ‘now is the time’  “for all good men” |
| TIMESTAMP | TIMESTAMP |  | 时间类型 |  |
| BINARY | BINARY |  | 字节数组 |  |

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。

## 3.2 集合数据类型

|  |  |  |
| --- | --- | --- |
| 数据类型 | 描述 | 语法示例 |
| STRUCT | 和c语言中的struct类似，都可以通过“点”符号访问元素内容。例如，如果某个列的数据类型是STRUCT{first STRING, last STRING},那么第1个元素可以通过字段.first来引用。2 | struct()  例如  struct<street:string, city:string> |
| MAP | MAP是一组键-值对元组集合，使用数组表示法可以访问数据。例如，如果某个列的数据类型是MAP，其中键->值对是’first’->’John’和’last’->’Doe’，那么可以通过字段名[‘last’]获取最后一个元素 | map()  例如map<string, int> |
| ARRAY | 数组是一组具有相同类型和名称的变量的集合。这些变量称为数组的元素，每个数组元素都有一个编号，编号从零开始。例如，数组值为[‘John’, ‘Doe’]，那么第2个元素可以通过数组名[1]进行引用。 | Array()  例如array<string> |

Hive有三种复杂数据类型ARRAY、MAP 和 STRUCT。ARRAY和MAP与Java中的Array和Map类似，而STRUCT与C语言中的Struct类似，它封装了一个命名字段集合，复杂数据类型允许任意层次的嵌套。

## 3.3 类型转化

Hive的原子数据类型是可以进行隐式转换的，类似于Java的类型转换，例如某表达式使用INT类型，TINYINT会自动转换为INT类型，但是Hive不会进行反向转化，例如，某表达式使用TINYINT类型，INT不会自动转换为TINYINT类型，它会返回错误，除非使用CAST操作。

**1）隐式类型转换规则如下**

（1）任何整数类型都可以隐式地转换为一个范围更广的类型，如TINYINT可以转换成INT，INT可以转换成BIGINT。

（2）所有整数类型、FLOAT和STRING类型都可以隐式地转换成DOUBLE。

（3）TINYINT、SMALLINT、INT都可以转换为FLOAT。

（4）BOOLEAN类型不可以转换为任何其它的类型。

**2）可以使用CAST操作显示进行数据类型转换**

例如CAST('1' AS INT)将把字符串'1' 转换成整数1；如果强制类型转换失败，如执行CAST('X' AS INT)，表达式返回空值 NULL。

0: jdbc:hive2://hadoop102:10000> select '1'+2, cast('1'as int) + 2;

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

| \_c0 | \_c1 |

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

| 3.0 | 3 |

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

# 第4章 DDL数据定义

## 4.1 创建数据库

CREATE DATABASE [IF NOT EXISTS] database\_name

[COMMENT database\_comment]

[LOCATION hdfs\_path]

[WITH DBPROPERTIES (property\_name=property\_value, ...)];

**1）创建一个数据库，数据库在HDFS上的默认存储路径是/user/hive/warehouse/\*.db。**

hive (default)> create database db\_hive;

**2）避免要创建的数据库已经存在错误，增加if not exists判断。（标准写法）**

hive (default)> create database db\_hive;

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

hive (default)> create database if not exists db\_hive;

**3）创建一个数据库，指定数据库在HDFS上存放的位置**

hive (default)> create database db\_hive2 location '/db\_hive2';

## 4.2 查询数据库

### 4.2.1 显示数据库

**1）显示数据库**

hive> show databases;

**2）过滤显示查询的数据库**

hive> show databases like 'db\_hive\*';

OK

db\_hive

db\_hive\_1

### 4.2.2 查看数据库详情

**1）显示数据库信息**

hive> desc database db\_hive;

OK

db\_hive hdfs://hadoop102:9000/user/hive/warehouse/db\_hive.db user1USER

**2）显示数据库详细信息，extended**

hive> desc database extended db\_hive;

OK

db\_hive hdfs://hadoop102:9000/user/hive/warehouse/db\_hive.db user1USER

### 4.2.3 切换当前数据库

hive (default)> use db\_hive;

## 4.3 修改数据库

用户可以使用ALTER DATABASE命令为某个数据库的DBPROPERTIES设置键-值对属性值，来描述这个数据库的属性信息。数据库的其他元数据信息都是不可更改的，包括数据库名和数据库所在的目录位置。

hive (default)> alter database db\_hive set dbproperties('createtime'='20170830');

在hive中查看修改结果

hive> desc database extended db\_hive;

db\_name comment location owner\_name owner\_type parameters

db\_hive hdfs://hadoop102:8020/user/hive/warehouse/db\_hive.db user1 USER {createtime=20170830}

## 4.4 删除数据库

**1）删除空数据库**

hive>drop database db\_hive2;

**2）如果删除的数据库不存在，最好采用 if exists判断数据库是否存在**

hive> drop database db\_hive;

FAILED: SemanticException [Error 10072]: Database does not exist: db\_hive

hive> drop database if exists db\_hive2;

**3）如果数据库不为空，可以采用cascade命令，强制删除**

hive> drop database db\_hive;

FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. InvalidOperationException(message:Database db\_hive is not empty. One or more tables exist.)

hive> drop database db\_hive cascade;

## 4.5 创建表

**1）建表语法**

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

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

[COMMENT table\_comment]

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

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

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

[ROW FORMAT row\_format]

[STORED AS file\_format]

[LOCATION hdfs\_path]

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

[AS select\_statement]

[LIKE table\_name]

**2）字段解释说明**

（1）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 选项来忽略这个异常。

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

（3）COMMENT：为表和列添加注释。

（4）PARTITIONED BY创建分区表

（5）CLUSTERED BY创建分桶表

（6）SORTED BY不常用，对桶中的一个或多个列另外排序

（7）ROW FORMAT

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

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

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

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

SerDe是Serialize/Deserilize的简称， hive使用Serde进行行对象的序列与反序列化。

（8）STORED AS指定存储文件类型

常用的存储文件类型：SEQUENCEFILE（二进制序列文件）、TEXTFILE（文本）、RCFILE（列式存储格式文件）

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

（9）LOCATION ：指定表在HDFS上的存储位置。

（10）AS：后跟查询语句，根据查询结果创建表。

（11）LIKE允许用户复制现有的表结构，但是不复制数据。

### 4.5.1 管理表(内部表)

**1）理论**

默认创建的表都是所谓的管理表，有时也被称为内部表。因为这种表，Hive会（或多或少地）控制着数据的生命周期。Hive默认情况下会将这些表的数据存储在由配置项hive.metastore.warehouse.dir(例如，/user/hive/warehouse)所定义的目录的子目录下。 当我们删除一个管理表时，Hive也会删除这个表中数据。管理表不适合和其他工具共享数据。

### 4.5.2 外部表

**1）理论**

因为表是外部表，所以Hive并非认为其完全拥有这份数据。删除该表并不会删除掉这份数据，不过描述表的元数据信息会被删除掉。

**2）管理表和外部表的使用场景**

每天将收集到的网站日志定期流入HDFS文本文件。在外部表（原始日志表）的基础上做大量的统计分析，用到的中间表、结果表使用内部表存储，数据通过SELECT+INSERT进入内部表。

### 4.5.3 管理表与外部表的互相转换

（1）查询表的类型

hive (default)> desc formatted student;

Table Type: MANAGED\_TABLE

（2）修改内部表student为外部表

alter table student set tblproperties('EXTERNAL'='TRUE');

（3）查询表的类型

hive (default)> desc formatted student;

Table Type: EXTERNAL\_TABLE

（4）修改外部表student为内部表

alter table student set tblproperties('EXTERNAL'='FALSE');

（5）查询表的类型

hive (default)> desc formatted student;

Table Type: MANAGED\_TABLE

注意：('EXTERNAL'='TRUE')和('EXTERNAL'='FALSE')为固定写法，区分大小写！

## 4.6 修改表

### 4.6.1 重命名表

**1）语法**

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

**2）实操案例**

hive (default)> alter table student2 rename to student3;

### 4.6.2 增加、修改和删除表分区

详见7.1.1章分区表基本操作。

### 4.6.3 增加/修改/替换列信息

**1）语法**

（1）更新列

更新列，列名可以随意修改，列的类型只能小改大，不能大改小（遵循自动转换规则）

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

（2）增加和替换列

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

注：ADD是代表新增一字段，字段位置在所有列后面(partition列前)，REPLACE则是表示替换表中所有字段，REPLACE使用的时候，字段的类型要跟之前的类型对应上，数量可以减少或者增加，其实就是包含了更新列，增加列，删除列的功能。

**2）实操案例**

（1）查询表结构

hive> desc dept;

（2）添加列

hive (default)> alter table dept add columns(deptdesc string);

（3）查询表结构

hive> desc dept;

（4）更新列

hive (default)> alter table dept change column deptdesc desc string;

（5）查询表结构

hive> desc dept;

（6）替换列

hive (default)> alter table dept replace columns(deptno string, dname

string, loc string);

（7）查询表结构

hive> desc dept;

## 4.7 删除表

hive (default)> drop table dept;

## 4.8 清除表

注意：Truncate只能删除管理表，不能删除外部表中数据

hive (default)> truncate table student;

# 第5章 DML数据操作

## 5.1 数据导入

### 5.1.1 向表中装载数据（Load）

**1）语法**

hive> load data [local] inpath '数据的path' [overwrite] into table student [partition (partcol1=val1,…)];

（1）load data:表示加载数据

（2）local:表示从本地加载数据到hive表；否则从HDFS加载数据到hive表

（3）inpath:表示加载数据的路径

（4）overwrite:表示覆盖表中已有数据，否则表示追加

（5）into table:表示加载到哪张表

（6）student:表示具体的表

（7）partition:表示上传到指定分区

**2）实操案例**

（0）创建一张表

hive (default)> create table student(id int, name string) row format delimited fields terminated by '\t';

（1）加载本地文件到hive

hive (default)> load data local inpath '/opt/module/hive/datas/student.txt' into table student;

（2）加载HDFS文件到hive中

上传文件到HDFS

hive (default)> dfs -put /opt/module/hive/datas/student.txt /user/user1;

加载HDFS上数据，导入完成后去hdfs查看文件是否还存在

hive (default)> load data inpath '/user/user1/student.txt' into table student;

（3）加载数据覆盖表中已有的数据

上传文件到HDFS

hive (default)> dfs -put /opt/module/hive/datas/student.txt /user/user1;

加载数据覆盖表中已有的数据

hive (default)> load data inpath '/user/user1/student.txt' overwrite into table student;

### 5.1.2 通过查询语句向表中插入数据（Insert）

**1）创建一张表**

hive (default)> create table student2(id int, name string) row format delimited fields terminated by '\t';

**2）基本模式插入数据**

hive (default)> insert into table student2 values(1,'wangwu'),(2,'zhaoliu');

**3）根据查询结果插入数据**

hive (default)> insert overwrite table student2

select id, name from student where id < 1006;

insert into：以追加数据的方式插入到表或分区，原有数据不会删除

insert overwrite：会覆盖表中已存在的数据

注意：insert不支持插入部分字段，并且后边跟select语句时，select之前不能加as，加了as会报错，一定要跟下面的as select区分开。

### 5.1.3 查询语句中创建表并加载数据（As Select）

根据查询结果创建表（查询的结果会添加到新创建的表中）

create table if not exists student3

as select id, name from student;

### 5.1.4 创建表时通过Location指定加载数据路径

**1）上传数据到hdfs上**

hive (default)> dfs -mkdir /student;

hive (default)> dfs -put /opt/module/hive/datas/student.txt /student;

**2）创建表，并指定在hdfs上的位置**

hive (default)> create external table if not exists student5(

id int, name string

)

row format delimited fields terminated by '\t'

location '/student';

**3）查询数据**

hive (default)> select \* from student5;

### 5.1.5 Import数据到指定Hive表中

注意：先用export导出后，再将数据导入。并且因为export导出的数据里面包含了元数据，因此import要导入的表不可以存在，否则报错。

hive (default)> import table student2 from

'/user/hive/warehouse/export/student';

## 5.2 数据导出

### 5.2.1 Insert导出

**1）将查询的结果导出到本地**

hive (default)> insert overwrite local directory '/opt/module/hive/datas/export/student'

select \* from student;

**2）将查询的结果格式化导出到本地**

hive(default)>insert overwrite local directory '/opt/module/hive/datas/export/student1'

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' select \* from student;

**3）将查询的结果导出到HDFS上(没有local)**

hive (default)> insert overwrite directory '/user/user1/student2'

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

select \* from student;

注意：insert 导出，导出的目录不用自己提前创建，hive会帮我们自动创建，但是由于是overwrite，所以导出路径一定要写具体，否则很可能会误删数据。这个步骤很重要，切勿大意。

### 5.2.2 Hadoop命令导出到本地

hive (default)> dfs -get /user/hive/warehouse/student/student.txt

/opt/module/hive/datas/export/student3.txt;

### 5.2.3 Hive Shell 命令导出

基本语法：（hive -f/-e 执行语句或者脚本 > file）

[user1@hadoop102 hive]$ bin/hive -e 'select \* from default.student;' >

/opt/module/hive/datas/export/student4.txt;

### 5.2.4 Export导出到HDFS上

hive (default)> export table default.student to

'/user/hive/warehouse/export/student';

export和import主要用于两个Hadoop平台集群之间Hive表迁移，不能直接导出的本地。

# 第6章 查询

查询语句语法：

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

  FROM table\_reference

  [WHERE where\_condition]

  [GROUP BY col\_list]

  [ORDER BY col\_list]

  [CLUSTER BY col\_list

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

  ]

 [LIMIT number]

## 6.1 基本查询（Select…From）

### 6.1.1 全表和特定列查询

**0）数据准备**

（0）原始数据

dept:

10 ACCOUNTING 1700

20 RESEARCH 1800

30 SALES 1900

40 OPERATIONS 1700

emp：

7369 SMITH CLERK 7902 1980-12-17 800.00 20

7499 ALLEN SALESMAN 7698 1981-2-20 1600.00 300.00 30

7521 WARD SALESMAN 7698 1981-2-22 1250.00 500.00 30

7566 JONES MANAGER 7839 1981-4-2 2975.00 20

7654 MARTIN SALESMAN 7698 1981-9-28 1250.00 1400.00 30

7698 BLAKE MANAGER 7839 1981-5-1 2850.00 30

7782 CLARK MANAGER 7839 1981-6-9 2450.00 10

7788 SCOTT ANALYST 7566 1987-4-19 3000.00 20

7839 KING PRESIDENT 1981-11-17 5000.00 10

7844 TURNER SALESMAN 7698 1981-9-8 1500.00 0.00 30

7876 ADAMS CLERK 7788 1987-5-23 1100.00 20

7900 JAMES CLERK 7698 1981-12-3 950.00 30

7902 FORD ANALYST 7566 1981-12-3 3000.00 20

7934 MILLER CLERK 7782 1982-1-23 1300.00 10

（1）创建部门表

create table if not exists dept(

deptno int,

dname string,

loc int

)

row format delimited fields terminated by '\t';

（2）创建员工表

（3）导入数据

load data local inpath '/opt/module/hive/datas/dept.txt' into table

dept;

load data local inpath '/opt/module/hive/datas/emp.txt' into table emp;

**1）全表查询**

hive (default)> select \* from emp;

**2）选择特定列查询**

hive (default)> select empno, ename from emp;

注意：

（1）SQL 语言大小写不敏感。

（2）SQL 可以写在一行或者多行

（3）关键字不能被缩写也不能分行

（4）各子句一般要分行写。

（5）使用缩进提高语句的可读性。

### 6.1.2 列别名

**1）重命名一个列**

**2）便于计算**

**3）紧跟列名，也可以在列名和别名之间加入关键字‘AS’**

**4）案例实操**

查询名称和部门

hive (default)> select ename AS name, deptno dn from emp;

### 6.1.3 算术运算符

|  |  |
| --- | --- |
| 运算符 | 描述 |
| A+B | A和B 相加 |
| A-B | A减去B |
| A\*B | A和B 相乘 |
| A/B | A除以B |
| A%B | A对B取余 |
| A&B | A和B按位取与 |
| A|B | A和B按位取或 |
| A^B | A和B按位取异或 |
| ~A | A按位取反 |

案例实操：查询出所有员工的薪水后加1显示。

hive (default)> select sal +1 from emp;

### 6.1.4 常用函数

**1）求总行数（count）**

hive (default)> select count(\*) cnt from emp;

**2）求工资的最大值（max）**

hive (default)> select max(sal) max\_sal from emp;

**3）求工资的最小值（min）**

hive (default)> select min(sal) min\_sal from emp;

**4）求工资的总和（sum）**

hive (default)> select sum(sal) sum\_sal from emp;

**5）求工资的平均值（avg）**

hive (default)> select avg(sal) avg\_sal from emp;

### 6.1.5 Limit语句

典型的查询会返回多行数据。LIMIT子句用于限制返回的行数。

hive (default)> select \* from emp limit 5;

hive (default)> select \* from emp limit 2,3;

### 6.1.6 Where语句

**1）使用WHERE子句，将不满足条件的行过滤掉**

**2）WHERE子句紧随FROM子句**

**3）案例实操**

查询出薪水大于1000的所有员工

hive (default)> select \* from emp where sal >1000;

注意：where子句中不能使用字段别名。

### 6.1.7 比较运算符（Between/In/ Is Null）

**1）下面表中描述了谓词操作符，这些操作符同样可以用于JOIN…ON和HAVING语句中。**

|  |  |  |
| --- | --- | --- |
| 操作符 | 支持的数据类型 | 描述 |
| A=B | 基本数据类型 | 如果A等于B则返回TRUE，反之返回FALSE |
| A<=>B | 基本数据类型 | 如果A和B都为NULL，则返回TRUE，如果一边为NULL，返回False |
| A<>B, A!=B | 基本数据类型 | A或者B为NULL则返回NULL；如果A不等于B，则返回TRUE，反之返回FALSE |
| A<B | 基本数据类型 | A或者B为NULL，则返回NULL；如果A小于B，则返回TRUE，反之返回FALSE |
| A<=B | 基本数据类型 | A或者B为NULL，则返回NULL；如果A小于等于B，则返回TRUE，反之返回FALSE |
| A>B | 基本数据类型 | A或者B为NULL，则返回NULL；如果A大于B，则返回TRUE，反之返回FALSE |
| A>=B | 基本数据类型 | A或者B为NULL，则返回NULL；如果A大于等于B，则返回TRUE，反之返回FALSE |
| A [NOT] BETWEEN B AND C | 基本数据类型 | 如果A，B或者C任一为NULL，则结果为NULL。如果A的值大于等于B而且小于或等于C，则结果为TRUE，反之为FALSE。如果使用NOT关键字则可达到相反的效果。 |
| A IS NULL | 所有数据类型 | 如果A等于NULL，则返回TRUE，反之返回FALSE |
| A IS NOT NULL | 所有数据类型 | 如果A不等于NULL，则返回TRUE，反之返回FALSE |
| IN(数值1, 数值2) | 所有数据类型 | 使用 IN运算显示列表中的值 |
| A [NOT] LIKE B | STRING 类型 | B是一个SQL下的简单正则表达式，也叫通配符模式，如果A与其匹配的话，则返回TRUE；反之返回FALSE。B的表达式说明如下：‘x%’表示A必须以字母‘x’开头，‘%x’表示A必须以字母’x’结尾，而‘%x%’表示A包含有字母’x’,可以位于开头，结尾或者字符串中间。如果使用NOT关键字则可达到相反的效果。 |
| A RLIKE B, A REGEXP B | STRING 类型 | B是基于java的正则表达式，如果A与其匹配，则返回TRUE；反之返回FALSE。匹配使用的是JDK中的正则表达式接口实现的，因为正则也依据其中的规则。例如，正则表达式必须和整个字符串A相匹配，而不是只需与其字符串匹配。 |

**2）案例实操**

（1）查询出薪水等于5000的所有员工

hive (default)> select \* from emp where sal =5000;

（2）查询工资在500到1000的员工信息

hive (default)> select \* from emp where sal between 500 and 1000;

（3）查询comm为空的所有员工信息

hive (default)> select \* from emp where comm is null;

（4）查询工资是1500或5000的员工信息

hive (default)> select \* from emp where sal IN (1500, 5000);

### 6.1.8 Like和RLike

**1）使用LIKE运算选择类似的值**

**2）选择条件可以包含字符或数字:**

% 代表零个或多个字符(任意个字符)。

\_ 代表一个字符。

**3）RLIKE子句**

RLIKE子句是Hive中这个功能的一个扩展，其可以通过Java的正则表达式这个更强大的语言来指定匹配条件。

**4）案例实操**

（1）查找名字以A开头的员工信息

hive (default)> select \* from emp where ename LIKE 'A%';

hive (default)> select \* from emp where ename RLIKE '^A';

（2）查找名字中第二个字母为A的员工信息

hive (default)> select \* from emp where ename LIKE '\_A%';

hive (default)> select \* from emp where ename RLIKE '^.A';

（3）查找名字中带有A的员工信息

hive (default)> select \* from emp where ename LIKE '%A%';

hive (default)> select \* from emp where ename RLIKE '[A]';

### 6.1.9 逻辑运算符（And/Or/Not）

|  |  |
| --- | --- |
| 操作符 | 含义 |
| AND | 逻辑并 |
| OR | 逻辑或 |
| NOT | 逻辑否 |

**1）案例实操**

（1）查询薪水大于1000，部门是30

hive (default)> select \* from emp where sal>1000 and deptno=30;

（2）查询薪水大于1000，或者部门是30

hive (default)> select \* from emp where sal>1000 or deptno=30;

（3）查询除了20部门和30部门以外的员工信息

hive (default)> select \* from emp where deptno not IN(30, 20);

## 6.2 分组

### 6.2.1 Group By语句

GROUP BY语句通常会和聚合函数一起使用，按照一个或者多个列队结果进行分组，然后对每个组执行聚合操作。

**1）案例实操：**

（1）计算emp表每个部门的平均工资

hive (default)> select t.deptno, avg(t.sal) avg\_sal from emp t group by t.deptno;

（2）计算emp每个部门中每个岗位的最高薪水

hive (default)> select t.deptno, t.job, max(t.sal) max\_sal from emp t group by

t.deptno, t.job;

### 6.2.2 Having语句

**1）having与where不同点**

（1）where后面不能写分组聚合函数，而having后面可以使用分组聚合函数。

（2）having只用于group by分组统计语句。

**2）案例实操**

（1）求每个部门的平均薪水大于2000的部门

求每个部门的平均工资

hive (default)> select deptno, avg(sal) from emp group by deptno;

求每个部门的平均薪水大于2000的部门

hive (default)> select deptno, avg(sal) avg\_sal from emp group by deptno having

avg\_sal > 2000;

## 6.3 Join语句

### 6.3.1 等值Join

Hive支持通常的SQL JOIN语句，但是只支持等值连接，不支持非等值连接。

**1）案例实操**

（1）根据员工表和部门表中的部门编号相等，查询员工编号、员工名称和部门名称；

hive (default)> select e.empno, e.ename, d.deptno, d.dname from emp e join dept d on e.deptno = d.deptno;

### 6.3.2 表的别名

**1）好处**

（1）使用别名可以简化查询。

（2）使用表名前缀可以提高执行效率。

**2）案例实操**

合并员工表和部门表

hive (default)> select e.empno, e.ename, d.deptno from emp e join dept d on e.deptno

= d.deptno;

### 6.3.3 内连接

内连接：只有进行连接的两个表中都存在与连接条件相匹配的数据才会被保留下来。

hive (default)> select e.empno, e.ename, d.deptno from emp e join dept d on e.deptno

= d.deptno;

### 6.3.4 左外连接

左外连接：JOIN操作符左边表中符合WHERE子句的所有记录将会被返回。

hive (default)> select e.empno, e.ename, d.deptno from emp e left join dept d on e.deptno = d.deptno;

### 6.3.5 右外连接

右外连接：JOIN操作符右边表中符合WHERE子句的所有记录将会被返回。

hive (default)> select e.empno, e.ename, d.deptno from emp e right join dept d on e.deptno = d.deptno;

### 6.3.6 满外连接

满外连接：将会返回所有表中符合WHERE语句条件的所有记录。如果任一表的指定字段没有符合条件的值的话，那么就使用NULL值替代。

hive (default)> select e.empno, e.ename, d.deptno from emp e full join dept d on e.deptno

= d.deptno;

### 6.3.7 多表连接

注意：连接 n个表，至少需要n-1个连接条件。例如：连接三个表，至少需要两个连接条件。

数据准备 在/opt/module/hive/datas/下：vim location.txt

1700 Beijing

1800 London

1900 Tokyo

**1）创建位置表**

create table if not exists location(

loc int,

loc\_name string

)

row format delimited fields terminated by '\t';

**2）导入数据**

hive (default)> load data local inpath '/opt/module/hive/datas/location.txt' into table location;

**3）多表连接查询**

hive (default)>SELECT e.ename, d.dname, l.loc\_name

FROM emp e

JOIN dept d

ON d.deptno = e.deptno

JOIN location l

ON d.loc = l.loc;

大多数情况下，Hive会对每对JOIN连接对象启动一个MapReduce任务。本例中会首先启动一个MapReduce job对表e和表d进行连接操作，然后会再启动一个MapReduce job将第一个MapReduce job的输出和表l;进行连接操作。

注意：为什么不是表d和表l先进行连接操作呢？这是因为Hive总是按照从左到右的顺序执行的。

优化：当对3个或者更多表进行join连接时，如果每个on子句都使用相同的连接键的话，那么只会产生一个MapReduce job。

### 6.3.8 笛卡尔积

**1）笛卡尔集会在下面条件下产生**

（1）省略连接条件

（2）连接条件无效

（3）所有表中的所有行互相连接

**2）案例实操**

hive (default)> select empno, dname from emp, dept;

## 6.4 排序

### 6.4.1 全局排序（Order By）

Order By：全局排序，只有一个Reducer

**1）使用 ORDER BY 子句排序**

ASC（ascend）: 升序（默认）

DESC（descend）: 降序

**2）ORDER BY 子句在SELECT语句的结尾**

**3）案例实操**

（1）查询员工信息按工资升序排列

hive (default)> select \* from emp order by sal;

（2）查询员工信息按工资降序排列

hive (default)> select \* from emp order by sal desc;

### 6.4.2 按照别名排序

按照员工薪水的2倍排序

hive (default)> select ename, sal\*2 twosal from emp order by twosal;

### 6.4.3 多个列排序

按照部门和工资升序排序

hive (default)> select ename, deptno, sal from emp order by deptno, sal ;

### 6.4.4 每个MapReduce内部排序（Sort By）

Sort By：对于大规模的数据集order by的效率非常低。在很多情况下，并不需要全局排序，此时可以使用**sort by**。

Sort by为每个reducer产生一个排序文件。每个Reducer内部进行排序，对全局结果集来说不是排序。

**1）设置reduce个数**

hive (default)> set mapreduce.job.reduces=3;

**2）查看设置reduce个数**

hive (default)> set mapreduce.job.reduces;

**3）根据部门编号降序查看员工信息**

hive (default)> select \* from emp sort by deptno desc;

**4）将查询结果导入到文件中（按照部门编号降序排序）**

hive (default)> insert overwrite local directory '/opt/module/hive/datas/sortby-result'

select \* from emp sort by deptno desc;

### 6.4.5 分区（Distribute By）

Distribute By： 在有些情况下，我们需要控制某个特定行应该到哪个reducer，通常是为了进行后续的聚集操作。**distribute by** 子句可以做这件事。**distribute by**类似MR中partition（自定义分区），进行分区，结合sort by使用。

对于distribute by进行测试，一定要分配多reduce进行处理，否则无法看到distribute by的效果。

**1）案例实操：**

（1）先按照部门编号分区，再按照员工编号降序排序。

hive (default)> set mapreduce.job.reduces=3;

hive (default)> insert overwrite local directory '/opt/module/hive/datas/distribute-result' select \* from emp distribute by deptno sort by empno desc;

注意：

* + distribute by的分区规则是根据分区字段的hash码与reduce的个数进行模除后，余数相同的分到一个区。
  + Hive要求DISTRIBUTE BY语句要写在SORT BY语句之前。
  + 演示完以后mapreduce.job.reduces的值要设置回-1，否则下面分区or分桶表load跑mr的时候有可能会报错

### 6.4.6 分区排序（Cluster By）

当distribute by和sort by字段相同时，可以使用cluster by方式。

cluster by除了具有distribute by的功能外还兼具sort by的功能。但是排序只能是升序排序，不能指定排序规则为ASC或者DESC。

（1）以下两种写法等价

hive (default)> select \* from emp cluster by deptno;

hive (default)> select \* from emp distribute by deptno sort by deptno;

注意：按照部门编号分区，不一定就是固定死的数值，可以是20号和30号部门分到一个分区里面去。

# 第7章 分区表和分桶表

## 7.1 分区表

分区表实际上就是对应一个HDFS文件系统上的独立的文件夹，该文件夹下是该分区所有的数据文件。Hive中的分区就是分目录，把一个大的数据集根据业务需要分割成小的数据集。在查询时通过WHERE子句中的表达式选择查询所需要的指定的分区，这样的查询效率会提高很多。

### 7.1.1 分区表基本操作

**1）引入分区表（需要根据日期对日志进行管理, 通过部门信息模拟）**

dept\_20200401.log

dept\_20200402.log

dept\_20200403.log

**2）创建分区表语法**

hive (default)> create table dept\_partition(

deptno int, dname string, loc string

)

partitioned by (day string)

row format delimited fields terminated by '\t';

|  |
| --- |
| 注意：分区字段不能是表中已经存在的数据，可以将分区字段看作表的伪列。 |

**3）加载数据到分区表中**

1. **数据准备**

dept\_20200401.log

10 ACCOUNTING 1700

20 RESEARCH 1800

dept\_20200402.log

30 SALES 1900

40 OPERATIONS 1700

dept\_20200403.log

50 TEST 2000

60 DEV 1900

1. **加载数据**

hive (default)> load data local inpath '/opt/module/hive/datas/dept\_20200401.log' into table dept\_partition partition(day='20200401');

hive (default)> load data local inpath '/opt/module/hive/datas/dept\_20200402.log' into table dept\_partition partition(day='20200402');

hive (default)> load data local inpath '/opt/module/hive/datas/dept\_20200403.log' into table dept\_partition partition(day='20200403');

注意：分区表加载数据时，必须指定分区

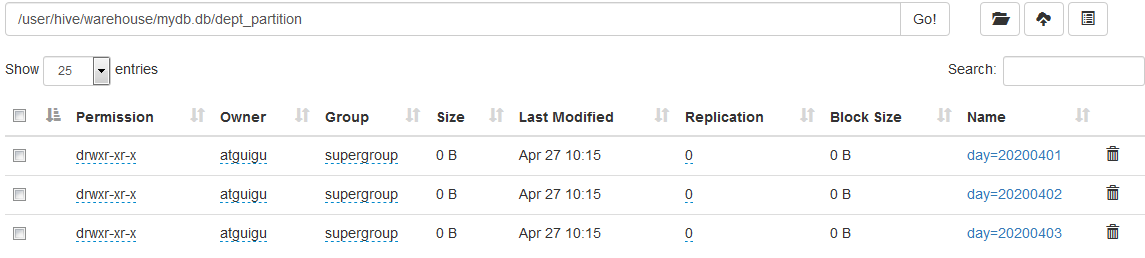


图 分区表

**4）查询分区表中数据**

单分区查询

hive (default)> select \* from dept\_partition where day='20200401';

多分区联合查询

hive (default)> select \* from dept\_partition where day='20200401'

union

select \* from dept\_partition where day='20200402'

union

select \* from dept\_partition where day='20200403';

hive (default)> select \* from dept\_partition where day='20200401' or

day='20200402' or day='20200403' ;

**5）查看分区表有多少分区**

hive> show partitions dept\_partition;

**6）增加分区**

创建单个分区

hive (default)> alter table dept\_partition add partition(day='20200404') ;

同时创建多个分区（分区之间不能有逗号）

hive (default)> alter table dept\_partition add partition(day='20200405') partition(day='20200406');

**7）删除分区**

删除单个分区

hive (default)> alter table dept\_partition drop partition (day='20200406');

同时删除多个分区（分区之间必须有逗号）

hive (default)> alter table dept\_partition drop partition (day='20200404'), partition(day='20200405');

**8）查看分区表结构**

hive> desc formatted dept\_partition;

# Partition Information

# col\_name data\_type comment

month string

### 7.1.2 分区表二级分区

思考: 如何一天的日志数据量也很大，如何再将数据拆分?

**1）创建二级分区表**

|  |
| --- |
| hive (default)> create table dept\_partition2(  deptno int, dname string, loc string  )  partitioned by (day string, hour string)  row format delimited fields terminated by '\t'; |

**2）正常的加载数据**

（1）加载数据到二级分区表中

hive (default)> load data local inpath '/opt/module/hive/datas/dept\_20200401.log' into table

dept\_partition2 partition(day='20200401', hour='12');

（2）查询分区数据

hive (default)> select \* from dept\_partition2 where day='20200401' and hour='12';

**3）把数据直接上传到分区目录上，让分区表和数据产生关联的三种方式**

（1）方式一：上传数据后修复

上传数据

hive (default)> dfs -mkdir -p

/user/hive/warehouse/db\_hive.db/dept\_partition2/day=20200401/hour=13;

hive (default)> dfs -put /opt/module/datas/dept\_20200401.log /user/hive/warehouse/db\_hive.db/dept\_partition2/day=20200401/hour=13;

查询数据（查询不到刚上传的数据）

hive (default)> select \* from dept\_partition2 where day='20200401' and hour='13';

执行修复命令

hive> msck repair table dept\_partition2;

再次查询数据

hive (default)> select \* from dept\_partition2 where day='20200401' and hour='13';

（2）方式二：上传数据后添加分区

上传数据

hive (default)> dfs -mkdir -p

/user/hive/warehouse/db\_hive.db/dept\_partition2/day=20200401/hour=14;

hive (default)> dfs -put /opt/module/hive/datas/dept\_20200401.log /user/hive/warehouse/db\_hive.db/dept\_partition2/day=20200401/hour=14;

执行添加分区

hive (default)> alter table dept\_partition2 add partition(day='20200401',hour='14');

查询数据

hive (default)> select \* from dept\_partition2 where day='20200401' and hour='14';

（3）方式三：创建文件夹后load数据到分区

创建目录

hive (default)> dfs -mkdir -p

/user/hive/warehouse/db\_hive.db/dept\_partition2/day=20200401/hour=15;

上传数据

hive (default)> load data local inpath '/opt/module/hive/datas/dept\_20200401.log' into table

dept\_partition2 partition(day='20200401',hour='15');

查询数据

hive (default)> select \* from dept\_partition2 where day='20200401' and hour='15';

### 7.1.3 动态分区调整

关系型数据库中，对分区表Insert数据时候，数据库自动会根据分区字段的值，将数据插入到相应的分区中，Hive中也提供了类似的机制，即动态分区(Dynamic Partition)，只不过，使用Hive的动态分区，需要进行相应的配置。

**1）开启动态分区参数设置**

（1）开启动态分区功能（默认true，开启）

hive.exec.dynamic.partition=true

（2）设置为非严格模式（动态分区的模式，默认strict，表示必须指定至少一个分区为静态分区，nonstrict模式表示允许所有的分区字段都可以使用动态分区。）

hive.exec.dynamic.partition.mode=nonstrict

（3）在所有执行MR的节点上，最大一共可以创建多少个动态分区。默认1000

hive.exec.max.dynamic.partitions=1000

（4）在每个执行MR的节点上，最大可以创建多少个动态分区。该参数需要根据实际的数据来设定。比如：源数据中包含了一年的数据，即day字段有365个值，那么该参数就需要设置成大于365，如果使用默认值100，则会报错。

hive.exec.max.dynamic.partitions.pernode=100

（5）整个MR Job中，最大可以创建多少个HDFS文件。默认100000

hive.exec.max.created.files=100000

（6）当有空分区生成时，是否抛出异常。一般不需要设置。默认false

hive.error.on.empty.partition=false

**2）案例实操**

需求：将dept表中的数据按照地区（loc字段），插入到目标表dept\_partition的相应分区中。

（1）创建目标分区表

hive (default)> create table dept\_partition\_dy(id int, name string) partitioned by (loc int) row format delimited fields terminated by '\t';

|  |
| --- |
|  |

（2）设置动态分区

set hive.exec.dynamic.partition.mode = nonstrict;

hive (default)> insert into table dept\_partition\_dy partition(loc) select deptno, dname, loc from dept;

（3）查看目标分区表的分区情况

hive (default)> show partitions dept\_partition\_dy;

思考：目标分区表是如何匹配到分区字段的？

## 7.2 分桶表

分区提供一个隔离数据和优化查询的便利方式。不过，并非所有的数据集都可形成合理的分区。对于一张表或者分区，Hive 可以进一步组织成桶，也就是更为细粒度的数据范围划分。

分桶是将数据集分解成更容易管理的若干部分的另一个技术。

分区针对的是数据的存储路径；分桶针对的是数据文件。

**1）先创建分桶表，通过直接导入数据文件的方式**

（1）数据准备

1001 ss1

1002 ss2

1003 ss3

1004 ss4

1005 ss5

1006 ss6

1007 ss7

1008 ss8

1009 ss9

1010 ss10

1011 ss11

1012 ss12

1013 ss13

1014 ss14

1015 ss15

1016 ss16

（2）创建分桶表

create table stu\_buck(id int, name string)

clustered by(id)

into 4 buckets

row format delimited fields terminated by '\t';

（3）查看表结构

hive (default)> desc formatted stu\_buck;

Num Buckets: 4

（4）导入数据到分桶表中（hive新版本load数据跑mr，因此要改用hdfs路径导数据）

hive (default)> load data local inpath '/opt/module/hive/datas/student.txt' into table stu\_buck;

（5）查看创建的分桶表中是否分成4个桶

查看hdfs(http://hadoop102:9870)

（6）查询分桶的数据

hive(default)> select \* from stu\_buck;

分桶规则：

根据结果可知：Hive的分桶采用对分桶字段的值进行哈希，然后除以桶的个数求余的方式决定该条记录存放在哪个桶当中

# 第8章 函数

## 8.1 系统内置函数

**1）查看系统自带的函数**

hive> show functions;

**2）显示自带的函数的用法**

hive> desc function upper;

**3）详细显示自带的函数的用法**

hive> desc function extended upper;

## 8.2 常用内置函数

### 8.2.1 空字段赋值

**1）函数说明**

NVL：给值为NULL的数据赋值，它的格式是NVL( value，default\_value)。它的功能是如果value为NULL，则NVL函数返回default\_value的值，否则返回value的值，如果两个参数都为NULL ，则返回NULL。

**2）数据准备：采用员工表**

**3）查询：如果员工的comm为NULL，则用-1代替**

hive (default)> select comm,nvl(comm, -1) from emp;

OK

comm \_c1

NULL -1.0

300.0 300.0

500.0 500.0

NULL -1.0

1400.0 1400.0

NULL -1.0

NULL -1.0

NULL -1.0

NULL -1.0

0.0 0.0

NULL -1.0

NULL -1.0

NULL -1.0

NULL -1.0

**4）查询：如果员工的comm为NULL，则用领导id代替**

hive (default)> select comm, nvl(comm,mgr) from emp;

OK

comm \_c1

NULL 7902.0

300.0 300.0

500.0 500.0

NULL 7839.0

1400.0 1400.0

NULL 7839.0

NULL 7839.0

NULL 7566.0

NULL NULL

0.0 0.0

NULL 7788.0

NULL 7698.0

NULL 7566.0

NULL 7782.0

### 8.2.2 CASE WHEN THEN ELSE END

1）按需求查询数据

select

dept\_id,

sum(case sex when '男' then 1 else 0 end) male\_count,

sum(case sex when '女' then 1 else 0 end) female\_count

from

emp\_sex

group by

dept\_id;

### 8.2.3 行转列

**1）相关函数说明**

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

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

注意：CONCAT\_WS must be "string or array<string>"

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

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

### 8.2.4 列转行

**1）函数说明**

Split(str, separator)：将字符串按照后面的分隔符切割，转换成字符array。

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

LATERAL VIEW

用法：LATERAL VIEW udtf(expression) tableAlias AS columnAlias

解释：lateral view用于和split, explode等UDTF一起使用，它能够将一行数据拆成多行数据，在此基础上可以对拆分后的数据进行聚合。

lateral view首先为原始表的每行调用UDTF，UTDF会把一行拆分成一或者多行，lateral view再把结果组合，产生一个支持别名表的虚拟表。

### 8.2.5 窗口函数（开窗函数）

**1）相关函数说明**

OVER()：指定分析函数工作的数据窗口大小，这个数据窗口大小可能会随着行的变而变化。

CURRENT ROW：当前行

n PRECEDING：往前n行数据

n FOLLOWING：往后n行数据

UNBOUNDED：无边界

UNBOUNDED PRECEDING 前无边界，表示从前面的起点，

UNBOUNDED FOLLOWING后无边界，表示到后面的终点

LAG(col,n,default\_val)：往前第n行数据

LEAD(col,n, default\_val)：往后第n行数据

FIRST\_VALUE (col,true/false)：当前窗口下的第一个值，第二个参数为true，跳过空值

LAST\_VALUE (col,true/false)：当前窗口下的最后一个值，第二个参数为true，跳过空值

NTILE(n)：把有序窗口的行分发到指定数据的组中，各个组有编号，编号从1开始，对于每一行，NTILE返回此行所属的组的编号。注意：n必须为int类型。

### 8.2.6 Rank

**1）函数说明**

RANK() 排序相同时会重复，总数不会变

DENSE\_RANK() 排序相同时会重复，总数会减少

ROW\_NUMBER() 会根据顺序计算

## 8.3 自定义函数

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

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

**3）**根据用户自定义函数类别分为以下三种：

（1）UDF（User-Defined-Function）

一进一出

（2）UDAF（User-Defined Aggregation Function）

用户自定义聚合函数，多进一出

类似于：count/max/min

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

用户自定义表生成函数，一进多出

如lateral view explode()

**4）**官方文档地址

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

**5）**编程步骤：

（1）继承Hive提供的类

org.apache.hadoop.hive.ql.udf.generic.GenericUDF

org.apache.hadoop.hive.ql.udf.generic.GenericUDTF;

（2）实现类中的抽象方法

（3）在hive的命令行窗口创建函数

添加jar

add jar linux\_jar\_path

创建function

create [temporary] function [dbname.]function\_name AS class\_name;

（4）在hive的命令行窗口删除函数

drop [temporary] function [if exists] [dbname.]function\_name;

## 8.4 自定义UDF函数

**0）需求:**

自定义一个UDF实现计算给定字符串的长度，例如：

hive(default)> select my\_len("abcd");

4

**1）创建一个Maven工程Hive**

**2）导入依赖**

<dependencies>

<dependency>

<groupId>org.apache.hive</groupId>

<artifactId>hive-exec</artifactId>

<version>3.1.2</version>

</dependency>

</dependencies>

**3）创建一个类**

package com.user1.hive;

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

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

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

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.primitive.PrimitiveObjectInspectorFactory;

/\*\*

\* 自定义UDF函数，需要继承GenericUDF类

\* 需求: 计算指定字符串的长度

\*/

public class MyStringLength extends GenericUDF {

/\*\*

\*

\* @param arguments 输入参数类型的鉴别器对象

\* @return 返回值类型的鉴别器对象

\* @throws UDFArgumentException

\*/

@Override

public ObjectInspector initialize(ObjectInspector[] arguments) throws UDFArgumentException {

// 判断输入参数的个数

if(arguments.length !=1){

throw new UDFArgumentLengthException("Input Args Length Error!!!");

}

// 判断输入参数的类型

if(!arguments[0].getCategory().equals(ObjectInspector.Category.PRIMITIVE)){

throw new UDFArgumentTypeException(0,"Input Args Type Error!!!");

}

//函数本身返回值为int，需要返回int类型的鉴别器对象

return PrimitiveObjectInspectorFactory.javaIntObjectInspector;

}

/\*\*

\* 函数的逻辑处理

\* @param arguments 输入的参数

\* @return 返回值

\* @throws HiveException

\*/

@Override

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

if(arguments[0].get() == null){

return 0 ;

}

return arguments[0].get().toString().length();

}

@Override

public String getDisplayString(String[] children) {

return "";

}

}

**4）创建临时函数**

**（1）打成jar包上传到服务器/opt/module/hive/datas/myudf.jar**

**（2）将jar包添加到hive的classpath，临时生效**

hive (default)> add jar /opt/module/hive/datas/myudf.jar;

**（3）创建临时函数与开发好的java class关联**

hive (default)> create temporary function my\_len as "com.user1.hive.MyStringLength";

**（4）即可在hql中使用自定义的临时函数**

hive (default)> select ename,my\_len(ename) ename\_len from emp;

**（5）删除临时函数**

hive (default)> drop temporary function my\_len;

**注意：临时函数只跟会话有关系，跟库没有关系。只要创建临时函数的会话不断，在当前会话下，任意一个库都可以使用，其他会话全都不能使用。**

**5）创建永久函数**

**（1）在$HIVE\_HOME下面创建auxlib目录**

[user1@hadoop102 hive]$ mkdir auxlib

**（2）将jar包上传到$HIVE\_HOME/auxlib下，然后重启hive**

**（3）创建永久函数**

hive (default)> create function my\_len2 as "com.user1.hive.MyStringLength";

**（4）即可在hql中使用自定义的永久函数**

hive (default)> select ename,my\_len2(ename) ename\_len from emp;

**（5）删除永久函数**

hive (default)> drop function my\_len2;

**注意：永久函数跟会话没有关系，创建函数的会话断了以后，其他会话也可以使用。**

**永久函数创建的时候，在函数名之前需要自己加上库名，如果不指定库名的话，会默认把当前库的库名给加上。**

**永久函数使用的时候，需要在指定的库里面操作,或者在其他库里面使用的话加上 库名.函数名**

# 第9章 压缩和存储

## 9.1 Hadoop压缩配置

### 9.1.1 MR支持的压缩编码

|  |  |  |  |
| --- | --- | --- | --- |
| 压缩格式 | 算法 | 文件扩展名 | 是否可切分 |
| DEFLATE | DEFLATE | .deflate | 否 |
| Gzip | DEFLATE | .gz | 否 |
| bzip2 | bzip2 | .bz2 | 是 |
| LZO | LZO | .lzo | 是 |
| Snappy | Snappy | .snappy | 否 |

为了支持多种压缩/解压缩算法，Hadoop引入了编码/解码器，如下表所示：

|  |  |
| --- | --- |
| 压缩格式 | 对应的编码/解码器 |
| DEFLATE | org.apache.hadoop.io.compress.DefaultCodec |
| gzip | org.apache.hadoop.io.compress.GzipCodec |
| bzip2 | org.apache.hadoop.io.compress.BZip2Codec |
| LZO | com.hadoop.compression.lzo.LzopCodec |
| Snappy | org.apache.hadoop.io.compress.SnappyCodec |

压缩性能的比较：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 压缩算法 | 原始文件大小 | 压缩文件大小 | 压缩速度 | 解压速度 |
| gzip | 8.3GB | 1.8GB | 17.5MB/s | 58MB/s |
| bzip2 | 8.3GB | 1.1GB | 2.4MB/s | 9.5MB/s |
| LZO | 8.3GB | 2.9GB | 49.3MB/s | 74.6MB/s |

<http://google.github.io/snappy/>

On a single core of a Core i7 processor in 64-bit mode, Snappy compresses at about 250 MB/sec or more and decompresses at about 500 MB/sec or more.

### 9.1.2 压缩参数配置

要在Hadoop中启用压缩，可以配置如下参数（mapred-site.xml文件中）：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 默认值 | 阶段 | 建议 |
| io.compression.codecs  （在core-site.xml中配置） | org.apache.hadoop.io.compress.DefaultCodec, org.apache.hadoop.io.compress.GzipCodec, org.apache.hadoop.io.compress.BZip2Codec,  org.apache.hadoop.io.compress.Lz4Codec | 输入压缩 | Hadoop使用文件扩展名判断是否支持某种编解码器 |
| mapreduce.map.output.compress | false | mapper输出 | 这个参数设为true启用压缩 |
| mapreduce.map.output.compress.codec | org.apache.hadoop.io.compress.DefaultCodec | mapper输出 | 使用LZO、LZ4或snappy编解码器在此阶段压缩数据 |
| mapreduce.output.fileoutputformat.compress | false | reducer输出 | 这个参数设为true启用压缩 |
| mapreduce.output.fileoutputformat.compress.codec | org.apache.hadoop.io.compress. DefaultCodec | reducer输出 | 使用标准工具或者编解码器，如gzip和bzip2 |
| mapreduce.output.fileoutputformat.compress.type | RECORD | reducer输出 | SequenceFile输出使用的压缩类型：NONE和BLOCK |

## 9.2 开启Map输出阶段压缩（MR引擎）

开启map输出阶段压缩可以减少job中map和Reduce task间数据传输量。具体配置如下：

**1）案例实操：**

（1）开启hive中间传输数据压缩功能

hive (default)>set hive.exec.compress.intermediate=true;

（2）开启mapreduce中map输出压缩功能

hive (default)>set mapreduce.map.output.compress=true;

（3）设置mapreduce中map输出数据的压缩方式

hive (default)>set mapreduce.map.output.compress.codec=org.apache.hadoop.io.compress.SnappyCodec;

（4）执行查询语句

hive (default)> select count(ename) name from emp;

## 9.3 开启Reduce输出阶段压缩

当Hive将输出写入到表中时，输出内容同样可以进行压缩。属性hive.exec.compress.output控制着这个功能。用户可能需要保持默认设置文件中的默认值false，这样默认的输出就是非压缩的纯文本文件了。用户可以通过在查询语句或执行脚本中设置这个值为true，来开启输出结果压缩功能。

**1）案例实操：**

（1）开启hive最终输出数据压缩功能

hive (default)>set hive.exec.compress.output=true;

（2）开启mapreduce最终输出数据压缩

hive (default)>set mapreduce.output.fileoutputformat.compress=true;

（3）设置mapreduce最终数据输出压缩方式

hive (default)> set mapreduce.output.fileoutputformat.compress.codec =org.apache.hadoop.io.compress.SnappyCodec;

（4）设置mapreduce最终数据输出压缩为块压缩

hive (default)> set mapreduce.output.fileoutputformat.compress.type=BLOCK;

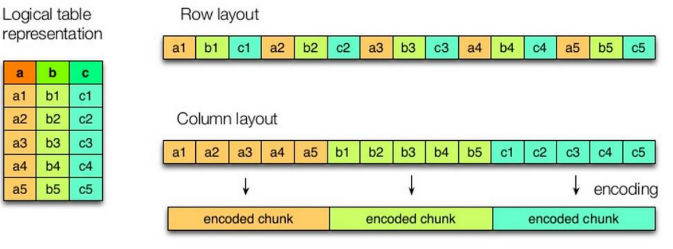
（5）测试一下输出结果是否是压缩文件

hive (default)> insert overwrite local directory '/opt/module/hive/datas/compress/' select \* from emp distribute by deptno sort by empno desc;

## 9.4 文件存储格式

Hive支持的存储数据的格式主要有：TEXTFILE 、SEQUENCEFILE、ORC、PARQUET。

### 9.4.1 列式存储和行式存储



如图所示左边为逻辑表，右边第一个为行式存储，第二个为列式存储。

**1）行存储的特点**

查询满足条件的一整行数据的时候，列存储则需要去每个聚集的字段找到对应的每个列的值，行存储只需要找到其中一个值，其余的值都在相邻地方，所以此时行存储查询的速度更快。

**2）列存储的特点**

因为每个字段的数据聚集存储，在查询只需要少数几个字段的时候，能大大减少读取的数据量；每个字段的数据类型一定是相同的，列式存储可以针对性的设计更好的设计压缩算法。

TEXTFILE和SEQUENCEFILE的存储格式都是基于行存储的；

ORC和PARQUET是基于列式存储的。

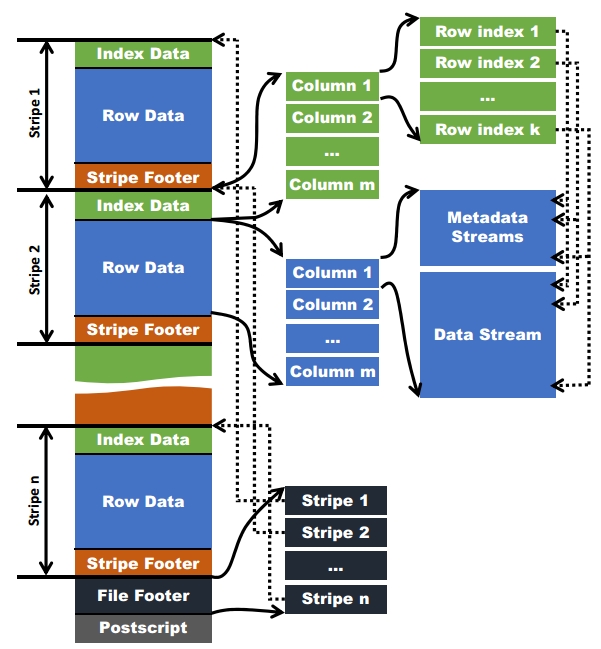
### 9.4.2 TextFile格式

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，但使用Gzip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。

### 9.4.3 Orc格式

Orc (Optimized Row Columnar)是Hive 0.11版里引入的新的存储格式。

如下图所示可以看到每个Orc文件由1个或多个stripe组成，每个stripe一般为HDFS的块大小，每一个stripe包含多条记录，这些记录按照列进行独立存储，对应到Parquet中的row group的概念。每个Stripe里有三部分组成，分别是Index Data，Row Data，Stripe Footer：



1）Index Data：一个轻量级的index，默认是每隔1W行做一个索引。这里做的索引应该只是记录某行的各字段在Row Data中的offset。

2）Row Data：存的是具体的数据，先取部分行，然后对这些行按列进行存储。对每个列进行了编码，分成多个Stream来存储。

3）Stripe Footer：存的是各个Stream的类型，长度等信息。

每个文件有一个File Footer，这里面存的是每个Stripe的行数，每个Column的数据类型信息等；每个文件的尾部是一个PostScript，这里面记录了整个文件的压缩类型以及FileFooter的长度信息等。在读取文件时，会seek到文件尾部读PostScript，从里面解析到File Footer长度，再读FileFooter，从里面解析到各个Stripe信息，再读各个Stripe，即从后往前读。

### 9.4.4 Parquet格式

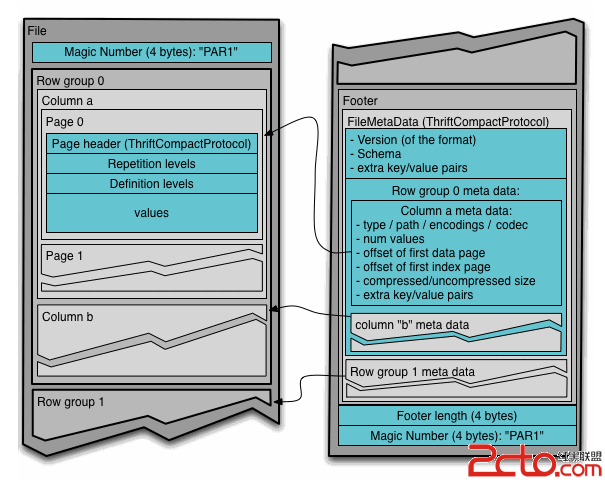
Parquet文件是以二进制方式存储的，所以是不可以直接读取的，文件中包括该文件的数据和元数据，因此Parquet格式文件是自解析的。

（1）行组(Row Group)：每一个行组包含一定的行数，在一个HDFS文件中至少存储一个行组，类似于orc的stripe的概念。

（2）列块(Column Chunk)：在一个行组中每一列保存在一个列块中，行组中的所有列连续的存储在这个行组文件中。一个列块中的值都是相同类型的，不同的列块可能使用不同的算法进行压缩。

（3）页(Page)：每一个列块划分为多个页，一个页是最小的编码的单位，在同一个列块的不同页可能使用不同的编码方式。

通常情况下，在存储Parquet数据的时候会按照Block大小设置行组的大小，由于一般情况下每一个Mapper任务处理数据的最小单位是一个Block，这样可以把每一个行组由一个Mapper任务处理，增大任务执行并行度。Parquet文件的格式。

2

上图展示了一个Parquet文件的内容，一个文件中可以存储多个行组，文件的首位都是该文件的Magic Code，用于校验它是否是一个Parquet文件，Footer length记录了文件元数据的大小，通过该值和文件长度可以计算出元数据的偏移量，文件的元数据中包括每一个行组的元数据信息和该文件存储数据的Schema信息。除了文件中每一个行组的元数据，每一页的开始都会存储该页的元数据，在Parquet中，有三种类型的页：数据页、字典页和索引页。数据页用于存储当前行组中该列的值，字典页存储该列值的编码字典，每一个列块中最多包含一个字典页，索引页用来存储当前行组下该列的索引，目前Parquet中还不支持索引页。

## 9.5 存储和压缩结合

### 9.5.1 测试存储和压缩

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

ORC存储方式的压缩：

|  |  |  |
| --- | --- | --- |
| Key | Default | Notes |
| orc.compress | ZLIB | high level compression (one of NONE, ZLIB, SNAPPY) |
| orc.compress.size | 262,144 | number of bytes in each compression chunk |
| orc.stripe.size | 268,435,456 | number of bytes in each stripe |
| orc.row.index.stride | 10,000 | number of rows between index entries (must be >= 1000) |
| orc.create.index | true | whether to create row indexes |
| orc.bloom.filter.columns | "" | comma separated list of column names for which bloom filter should be created |
| orc.bloom.filter.fpp | 0.05 | false positive probability for bloom filter (must >0.0 and <1.0) |

注意：所有关于ORCFile的参数都是在HQL语句的TBLPROPERTIES字段里面出现

**1）创建一个ZLIB压缩的ORC存储方式**

（1）建表语句

create table log\_orc\_zlib(

track\_time string,

url string,

session\_id string,

referer string,

ip string,

end\_user\_id string,

city\_id string

)

row format delimited fields terminated by '\t'

stored as orc

tblproperties("orc.compress"="ZLIB");

（2）插入数据

insert into log\_orc\_zlib select \* from log\_text;

|  |
| --- |
|  |

（3）查看插入后数据

hive (default)> dfs -du -h /user/hive/warehouse/log\_orc\_zlib/ ;

|  |
| --- |
|  |

2.78 M /user/hive/warehouse/log\_orc\_none/000000\_0

**2）创建一个SNAPPY压缩的ORC存储方式**

（1）建表语句

create table log\_orc\_snappy(

track\_time string,

url string,

session\_id string,

referer string,

ip string,

end\_user\_id string,

city\_id string

)

row format delimited fields terminated by '\t'

stored as orc

tblproperties("orc.compress"="SNAPPY");

|  |
| --- |
|  |

（2）插入数据

insert into log\_orc\_snappy select \* from log\_text;

|  |
| --- |
|  |

（3）查看插入后数据

hive (default)> dfs -du -h /user/hive/warehouse/log\_orc\_snappy/ ;

3.75 M /user/hive/warehouse/log\_orc\_snappy/000000\_0

ZLIB比Snappy压缩的还小。原因是ZLIB采用的是deflate压缩算法。比snappy压缩的压缩率高。

**3）创建一个SNAPPY压缩的parquet存储方式**

（1）建表语句

create table log\_parquet\_snappy(

track\_time string,

url string,

session\_id string,

referer string,

ip string,

end\_user\_id string,

city\_id string

)

row format delimited fields terminated by '\t'

stored as parquet

tblproperties("parquet.compression"="SNAPPY");

|  |
| --- |
|  |

（2）插入数据

insert into log\_parquet\_snappy select \* from log\_text;

|  |
| --- |
|  |

（3）查看插入后数据

hive (default)> dfs -du -h /user/hive/warehouse/log\_parquet\_snappy / ;

6.39 MB /user/hive/warehouse/ log\_parquet\_snappy /000000\_0

**4）存储方式和压缩总结**

在实际的项目开发当中，hive表的数据存储格式一般选择：orc或parquet。压缩方式一般选择snappy，lzo。