# Hive基本概念

## 什么是Hive

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。

本质是：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

### 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。
4. Hive优势在于处理大数据，对于处理小数据没有优势，因为Hive的执行延迟比较高。
5. 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。

### 缺点

1．Hive的HQL表达能力有限

（1）迭代式算法无法表达

（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自动生成的MapReduce作业，通常情况下不够智能化

（2）Hive调优比较困难，粒度较粗

## Hive架构原理



图6-1 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。

## Hive和数据库比较

由于 Hive 采用了类似SQL 的查询语言 HQL(Hive Query Language)，因此很容易将 Hive 理解为数据库。其实从结构上来看，Hive 和数据库除了拥有类似的查询语言，再无类似之处。本文将从多个方面来阐述 Hive 和数据库的差异。数据库可以用在 Online 的应用中，但是Hive 是为数据仓库而设计的，清楚这一点，有助于从应用角度理解 Hive 的特性。

### 查询语言

由于SQL被广泛的应用在数据仓库中，因此，专门针对Hive的特性设计了类SQL的查询语言HQL。熟悉SQL开发的开发者可以很方便的使用Hive进行开发。

### 数据存储位置

Hive 是建立在 Hadoop 之上的，所有 Hive 的数据都是存储在 HDFS 中的。而数据库则可以将数据保存在块设备或者本地文件系统中。

### 数据更新

由于Hive是针对数据仓库应用设计的，而数据仓库的内容是读多写少的。因此，Hive中不建议对数据的改写，所有的数据都是在加载的时候确定好的。而数据库中的数据通常是需要经常进行修改的，因此可以使用 INSERT INTO …  VALUES 添加数据，使用 UPDATE … SET修改数据。

### 执行

Hive中大多数查询的执行是通过 Hadoop 提供的 MapReduce 来实现的。而数据库通常有自己的执行引擎。

### 执行延迟

Hive 在查询数据的时候，由于没有索引，需要扫描整个表，因此延迟较高。另外一个导致 Hive 执行延迟高的因素是 MapReduce框架。由于MapReduce 本身具有较高的延迟，因此在利用MapReduce 执行Hive查询时，也会有较高的延迟。相对的，数据库的执行延迟较低。当然，这个低是有条件的，即数据规模较小，当数据规模大到超过数据库的处理能力的时候，Hive的并行计算显然能体现出优势。

### 可扩展性

由于Hive是建立在Hadoop之上的，因此Hive的可扩展性是和Hadoop的可扩展性是一致的（世界上最大的Hadoop 集群在 Yahoo!，2009年的规模在4000 台节点左右）。而数据库由于 ACID 语义的严格限制，扩展行非常有限。目前最先进的并行数据库 [Oracle](http://lib.csdn.net/base/oracle" \t "_blank" \o "Oracle知识库) 在理论上的扩展能力也只有100台左右。

### 数据规模

由于Hive建立在集群上并可以利用MapReduce进行并行计算，因此可以支持很大规模的数据；对应的，数据库可以支持的数据规模较小。

# Hive安装

## Hive安装地址

1．Hive官网地址

http://hive.apache.org/

2．文档查看地址

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

3．下载地址

http://archive.apache.org/dist/hive/

4．github地址

https://github.com/apache/hive

## MySql安装

### 安装包准备

1．卸载自带的Mysql-libs（如果之前安装过mysql，要全都卸载掉）

rpm -qa | grep -i -E mysql\|mariadb | xargs -n1 sudo rpm -e --nodeps

2．将安装包和JDBC驱动上传到/opt/software，共计6个

01\_mysql-community-common-5.7.29-1.el7.x86\_64.rpm

02\_mysql-community-libs-5.7.29-1.el7.x86\_64.rpm

03\_mysql-community-libs-compat-5.7.29-1.el7.x86\_64.rpm

04\_mysql-community-client-5.7.29-1.el7.x86\_64.rpm

05\_mysql-community-server-5.7.29-1.el7.x86\_64.rpm

mysql-connector-java-5.1.48.jar

### 安装MySql

1．安装mysql依赖

sudo rpm -ivh 01\_mysql-community-common-5.7.29-1.el7.x86\_64.rpm

sudo rpm -ivh 02\_mysql-community-libs-5.7.29-1.el7.x86\_64.rpm

sudo rpm -ivh 03\_mysql-community-libs-compat-5.7.29-1.el7.x86\_64.rpm

2．安装mysql-client

sudo rpm -ivh 04\_mysql-community-client-5.7.29-1.el7.x86\_64.rpm

3．安装mysql-server

sudo rpm -ivh 05\_mysql-community-server-5.7.29-1.el7.x86\_64.rpm

4．启动mysql

sudo systemctl start mysqld

5. 查看mysql密码

sudo cat /var/log/mysqld.log | grep password

### 配置MySql

配置只要是root用户+密码，在任何主机上都能登录MySQL数据库。

1．用刚刚查到的密码进入mysql（如果报错，给密码加单引号）

mysql -uroot -p’password’

2．设置复杂密码(由于mysql密码策略，此密码必须足够复杂)

set password=password("Qs23=zs32");

3．更改mysql密码策略

set global validate\_password\_length=4;

set global validate\_password\_policy=0;

4．设置简单好记的密码

set password=password("000000");

5．进入msyql库

use mysql

6．查询user表

select user, host from user;

7．修改user表，把Host表内容修改为%

update user set host="%" where user="root";

8．刷新

flush privileges;

10．退出

quit;

## Hive安装部署

1．Hive安装及配置

* + 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

tar -zxvf /opt/software/apache-hive-3.1.2-bin.tar.gz -C /opt/module/

* + 1. 修改apache-hive-3.1.2-bin.tar.gz的名称为hive

mv /opt/module/apache-hive-3.1.2-bin/ /opt/module/hive

* + 1. 修改/etc/profile.d/my\_env.sh，添加环境变量

sudo vim /etc/profile.d/my\_env.sh

添加内容

#HIVE\_HOME

export HIVE\_HOME=/opt/module/hive

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

重启Xshell对话框使环境变量生效

替换hive中的guava.jar

cp $HADOOP\_HOME/share/hadoop/common/lib/guava-27.0-jre.jar $HIVE\_HOME/lib/

rm $HIVE\_HOME/lib/guava-19.0.jar

* + 1. 解决日志Jar包冲突

mv $HIVE\_HOME/lib/log4j-slf4j-impl-2.10.0.jar $HIVE\_HOME/lib/log4j-slf4j-impl-2.10.0.bak

## Hive元数据配置到MySql

### 拷贝驱动

将MySQL的JDBC驱动拷贝到Hive的lib目录下

cp /opt/software/mysql-connector-java-5.1.48.jar $HIVE\_HOME/lib

### 配置Metastore到MySql

在$HIVE\_HOME/conf目录下新建hive-site.xml文件

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>

<property>

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

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

</property>

<property>

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

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

</property>

<property>

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

<value>root</value>

</property>

<property>

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

<value>000000</value>

</property>

<property>

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

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

</property>

<property>

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

<value>false</value>

</property>

<property>

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

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

</property>

<property>

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

<value>10000</value>

</property>

<property>

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

<value>hadoop102</value>

</property>

<property>

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

<value>false</value>

</property>

</configuration>

## 启动Hive

### 初始化元数据库

登陆MySQL

mysql -uroot -p000000

新建Hive元数据库

create database metastore;

quit;

初始化Hive元数据库

schematool -initSchema -dbType mysql -verbose

### 启动metastore和hiveserver2

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

编写hive服务启动脚本

vim $HIVE\_HOME/bin/hiveservices.sh

内容如下：

#!/bin/bash

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

META\_PID=/tmp/meta.pid

SERVER\_PID=/tmp/server.pid

function hive\_start()

{

hdfs dfsadmin -safemode wait >/dev/null 2>&1

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

echo $! > $META\_PID

sleep 4

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

echo $! > $SERVER\_PID

}

function hive\_stop()

{

if [ -f $META\_PID ]

then

cat $META\_PID | xargs kill

rm $META\_PID

else

echo "Meta PID文件丢失，请手动关闭服务"

fi

if [ -f $SERVER\_PID ]

then

cat $SERVER\_PID | xargs kill

rm $SERVER\_PID

else

echo "Server2 PID文件丢失，请手动关闭服务"

fi

}

case $1 in

"start")

hive\_start

;;

"stop")

hive\_stop

;;

"restart")

hive\_stop

sleep 2

hive\_start

;;

\*)

echo Invalid Args!

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

;;

esac

添加执行权限

chmod +x $HIVE\_HOME/bin/hiveservices.sh

启动Hive后台服务

hiveservices.sh start

## Hive常用交互命令

|  |
| --- |
| [atguigu@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) |

1．“-e”不进入hive的交互窗口执行sql语句

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

2．“-f”执行脚本中sql语句

1. 在/opt/module/datas目录下创建hivef.sql文件

[atguigu@hadoop102 datas]$ touch hivef.sql

文件中写入正确的sql语句

select \*from student;

1. 执行文件中的sql语句

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

1. 执行文件中的sql语句并将结果写入文件中

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

## Hive其他命令操作

1．退出hive窗口：

hive(default)>exit;

hive(default)>quit;

2．在hive cli命令窗口中如何查看hdfs文件系统

hive(default)>dfs -ls /;

3．在hive cli命令窗口中如何查看本地文件系统

hive(default)>! ls /opt/module/datas;

4．查看在hive中输入的所有历史命令

（1）进入到当前用户的根目录/root

（2）查看. hivehistory文件

[atguigu@hadoop102 ~]$ cat .hivehistory

## Hive常见属性配置

### Hive运行日志信息配置

1．Hive的log默认存放在/tmp/aima/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/opt/module/hive/conf/hive-log4j.properties.template文件名称为

hive-log4j.properties

[atguigu@hadoop102 conf]$ pwd

/opt/module/hive/conf

[atguigu@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

# Hive数据类型

## 基本数据类型

表6-1

|  |  |  |  |
| --- | --- | --- | --- |
| Hive数据类型 | Java数据类型 | 长度 | 例子 |
| TINYINT | byte | 1byte有符号整数 | 20 |
| SMALINT | short | 2byte有符号整数 | 20 |
| INT | int | 4byte有符号整数 | 20 |
| BIGINT | long | 8byte有符号整数 | 20 |
| BOOLEAN | boolean | 布尔类型，true或者false | TRUE FALSE |
| FLOAT | float | 单精度浮点数 | 3.14159 |
| DOUBLE | double | 双精度浮点数 | 3.14159 |
| STRING | 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 |  | 时间类型 |  |
| 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。

## 集合数据类型

表6-2

|  |  |  |
| --- | --- | --- |
| 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。 | 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。

案例实操

1. 假设某表有如下一行，我们用JSON格式来表示其数据结构。在Hive下访问的格式为

|  |
| --- |
| {  "name": "songsong",  "friends": ["bingbing" , "lili"] , //列表Array,  "children": { //键值Map,  "xiao song": 18 ,  "xiaoxiao song": 19  }  "address": { //结构Struct,  "street": "hui long guan" ,  "city": "beijing"  }  } |

2）基于上述数据结构，我们在Hive里创建对应的表，并导入数据。

创建本地测试文件test.txt

|  |
| --- |
| songsong,bingbing\_lili,xiao song:18\_xiaoxiao song:19,hui long guan\_beijing  yangyang,caicai\_susu,xiao yang:18\_xiaoxiao yang:19,chao yang\_beijing |

注意：MAP，STRUCT和ARRAY里的元素间关系都可以用同一个字符表示，这里用“\_”。

3）Hive上创建测试表test

|  |
| --- |
| create table test(  name string,  friends array<string>,  children map<string, int>,  address struct<street:string, city:string>  )  row format delimited fields terminated by ','  collection items terminated by '\_'  map keys terminated by ':'  lines terminated by '\n'; |

字段解释：

row format delimited fields terminated by ',' -- 列分隔符

collection items terminated by '\_' --MAP STRUCT 和 ARRAY 的分隔符(数据分割符号)

map keys terminated by ':' -- MAP中的key与value的分隔符

lines terminated by '\n'; -- 行分隔符

4）导入文本数据到测试表

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

5）访问三种集合列里的数据，以下分别是ARRAY，MAP，STRUCT的访问方式

|  |
| --- |
| hive (default)> select friends[1],children['xiao song'],address.city from test  where name="songsong";  OK  \_c0 \_c1 city  lili 18 beijing  Time taken: 0.076 seconds, Fetched: 1 row(s) |

# DDL数据定义

## 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.db';

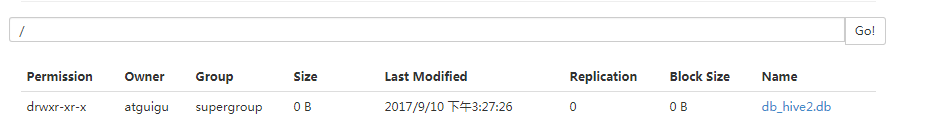


图6-4 数据库存放位置

## 查询数据库

### 显示数据库

1．显示数据库

hive> show databases;

### 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 atguiguUSER

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 atguiguUSER

### 切换当前数据库

hive (default)> use db\_hive;

## 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 atguigu USER {createtime=20170830}

## 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;

## 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]

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。

### 管理表

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。

2．案例实操

（1）普通创建表

|  |
| --- |
| create table if not exists student2(  id int, name string  )  row format delimited fields terminated by '\t'  stored as textfile  location '/user/hive/warehouse/student2'; |

（2）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; |

（3）查询表的类型

hive (default)> desc formatted student2;

Table Type: MANAGED\_TABLE

### 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．案例实操

分别创建部门和员工外部表，并向表中导入数据。

1. 上传数据到HDFS

hive (default)> dfs -mkdir /student;

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

（2）建表语句

创建外部表

|  |
| --- |
| hive (default)> create external table stu\_external(  id int,  name string)  row format delimited fields terminated by '\t'  location '/student'; |

（3）查看创建的表

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

OK

stu\_external.id stu\_external.name

1001 lisi

1002 wangwu

1003 zhaoliu

（4）查看表格式化数据

hive (default)> desc formatted dept;

Table Type: EXTERNAL\_TABLE

（5）删除外部表

hive (default)> drop table stu\_external;

外部表删除后，hdfs中的数据还在，但是metadata中stu\_external的元数据已被删除

## 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。

### 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）

/user/hive/warehouse/log\_partition/20170702/20170702.log

/user/hive/warehouse/log\_partition/20170703/20170703.log

/user/hive/warehouse/log\_partition/20170704/20170704.log

2．创建分区表语法

|  |
| --- |
| hive (default)> create table dept\_partition(  deptno int, dname string, loc string  )  partitioned by (month 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

hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table default.dept\_partition partition(month='202009');

hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table default.dept\_partition partition(month='202008');

hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table default.dept\_partition partition(month='202007');

注意：分区表加载数据时，必须指定分区

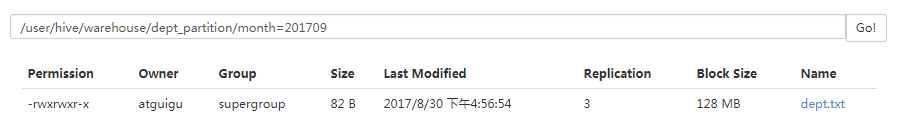


图6-5 加载数据到分区表

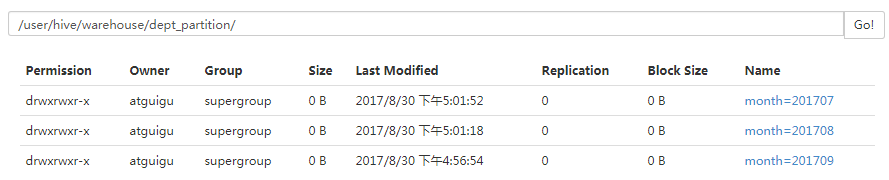


图6-6 分区表

4．查询分区表中数据

单分区查询

hive (default)> select \* from dept\_partition where month='202009';

多分区联合查询

hive (default)> select \* from dept\_partition where month='202009'

Union

select \* from dept\_partition where month='202008'

Union

select \* from dept\_partition where month='202007';

\_u3.deptno \_u3.dname \_u3.loc \_u3.month

10 ACCOUNTING NEW YORK 201707

10 ACCOUNTING NEW YORK 201708

10 ACCOUNTING NEW YORK 201709

20 RESEARCH DALLAS 202007

20 RESEARCH DALLAS 202008

20 RESEARCH DALLAS 202009

30 SALES CHICAGO 202007

30 SALES CHICAGO 201708

30 SALES CHICAGO 201709

40 OPERATIONS BOSTON 201707

40 OPERATIONS BOSTON 201708

40 OPERATIONS BOSTON 201709

5．增加分区

创建单个分区

hive (default)> alter table dept\_partition add partition(month='201706') ;

同时创建多个分区

hive (default)> alter table dept\_partition add partition(month='201705') partition(month='201704');

6．删除分区

删除单个分区

hive (default)> alter table dept\_partition drop partition (month='201704');

同时删除多个分区

hive (default)> alter table dept\_partition drop partition (month='201705'), partition (month='201706');

7．查看分区表有多少分区

hive> show partitions dept\_partition;

8．查看分区表结构

hive> desc formatted dept\_partition;

# Partition Information

# col\_name data\_type comment

month string

### 分区表注意事项

1．创建二级分区表

|  |
| --- |
| hive (default)> create table dept\_partition2(  deptno int, dname string, loc string  )  partitioned by (month string, day 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/datas/dept.txt' into table

default.dept\_partition2 partition(month='201709', day='13');

（2）查询分区数据

hive (default)> select \* from dept\_partition2 where month='201709' and day='13';

## 删除表

hive (default)> drop table dept\_partition;

# DML数据操作

## 数据导入

### 向表中装载数据（Load）

1．语法

hive> load data [local] inpath '/opt/module/datas/student.txt' [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 string, 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/datas/student.txt' into table default.student;

（2）加载HDFS文件到hive中

上传文件到HDFS

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

加载HDFS上数据

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

### 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/datas/student.txt /student;

2. 创建表，并指定在hdfs上的位置

hive (default)> create external table stu\_external(

id int,

name string)

row format delimited fields terminated by '\t'

location '/student';

3．查询数据

hive (default)> select \* from student5;

## 数据导出

### Insert导出

1．将查询的结果导出到本地

hive (default)> insert overwrite local directory '/opt/module/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/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/student2'

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

select \* from student;

# 查询

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

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] |

## 基本查询（Select…From）

### 全表和特定列查询

创建部门表

|  |
| --- |
| create table if not exists dept(  deptno int COMMENT'部门编号',  dname string COMMENT'部门名称',  loc int  )  row format delimited fields terminated by '\t'; |

创建员工表

|  |
| --- |
| create table if not exists emp(  empno int COMMENT '员工编号',  ename string COMMENT '员工名称',  job string COMMENT '员工职位',  mgr int,  hiredate string COMMENT '入职日期',  sal double COMMENT '工资',  comm double ,  deptno int COMMENT '所属部门编号')  row format delimited fields terminated by '\t'; |

导入数据

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

dept;

load data local inpath '/opt/module/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。

### 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;

### 算术运算符

表6-3

|  |  |
| --- | --- |
| 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;

### 常用函数

1．求总行数（count）

2．求工资的最大值（max）

3．求工资的最小值（min）

4．求工资的总和（sum）

5．求工资的平均值（avg）

hive (default)> select count(\*),max(sal),min(sal),sum(sal),avg(sal) from emp;

### 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;

## 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。

### 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。

表6-4

|  |  |  |
| --- | --- | --- |
| 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);

### 逻辑运算符（And/Or/Not）

表6-5

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

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);

## 分组

### 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。

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;

### Having语句

1．having与where不同点

（1）where后面不能写分组函数，而having后面可以使用分组函数。

（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;

## Join语句

### 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。

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;

### 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;

### 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;

### 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;

### 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;

### 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;

### 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



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/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。

## 排序

### 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/datas/sortby-result'

select \* from emp sort by deptno desc;

### 分区排序（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。

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/datas/distribute-result' select \* from emp distribute by deptno sort by empno desc;

注意：

1. 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。
2. Hive要求DISTRIBUTE BY语句要写在SORT BY语句之前。

### Cluster By

当distribute by和sorts 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。

## 其他常用查询函数

### 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。

1. 数据准备：采用员工表
2. 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

1. 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

### CASE WHEN

1. 数据准备

|  |  |  |
| --- | --- | --- |
| name | dept\_id | sex |
| 悟空 | A | 男 |
| 大海 | A | 男 |
| 宋宋 | B | 男 |
| 凤姐 | A | 女 |
| 婷姐 | B | 女 |
| 婷婷 | B | 女 |

2．需求

求出不同部门男女各多少人。结果如下：

A 2 1

B 1 2

3．创建本地emp\_sex.txt，导入数据

[atguigu@hadoop102 datas]$ vi emp\_sex.txt

悟空 A 男

大海 A 男

宋宋 B 男

凤姐 A 女

婷姐 B 女

婷婷 B 女

4．创建hive表并导入数据

create table emp\_sex(

name string,

dept\_id string,

sex string)

row format delimited fields terminated by "\t";

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

5．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; |

### 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;

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

2．数据准备

表6-6 数据准备

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

3．需求

把星座和血型一样的人归类到一起。结果如下：

射手座,A 大海|凤姐

白羊座,A 孙悟空|猪八戒

白羊座,B 宋宋|苍老师

4．创建本地constellation.txt，导入数据

[atguigu@hadoop102 datas]$ vi constellation.txt

孙悟空 白羊座 A

大海 射手座 A

宋宋 白羊座 B

猪八戒 白羊座 A

凤姐 射手座 A

5．创建hive表并导入数据

create table person\_info(

name string,

constellation string,

blood\_type string)

row format delimited fields terminated by "\t";

load data local inpath "/opt/module/datas/constellation.txt" into table person\_info;

6．按需求查询数据

|  |
| --- |
| select  t1.base,  concat\_ws('|', collect\_set(t1.name)) name  from  (select  name,  concat(constellation, ",", blood\_type) base  from  person\_info) t1  group by  t1.base; |

### 列转行

1．函数说明

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

LATERAL VIEW

用法：LATERAL VIEW udtf(expression) tableAlias AS columnAlias

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

2．数据准备

表6-7 数据准备

|  |  |
| --- | --- |
| movie | category |
| 《疑犯追踪》 | 悬疑,动作,科幻,剧情 |
| 《Lie to me》 | 悬疑,警匪,动作,心理,剧情 |
| 《战狼2》 | 战争,动作,灾难 |

3．需求

将电影分类中的数组数据展开。结果如下：

《疑犯追踪》 悬疑

《疑犯追踪》 动作

《疑犯追踪》 科幻

《疑犯追踪》 剧情

《Lie to me》 悬疑

《Lie to me》 警匪

《Lie to me》 动作

《Lie to me》 心理

《Lie to me》 剧情

《战狼2》 战争

《战狼2》 动作

《战狼2》 灾难

4．创建本地movie.txt，导入数据

[atguigu@hadoop102 datas]$ vi movie.txt

《疑犯追踪》 悬疑,动作,科幻,剧情

《Lie to me》 悬疑,警匪,动作,心理,剧情

《战狼2》 战争,动作,灾难

5．创建hive表并导入数据

|  |
| --- |
| create table movie\_info(  movie string,  category string)  row format delimited fields terminated by "\t";  load data local inpath "/opt/module/datas/movie.txt" into table movie\_info; |

6．按需求查询数据

|  |
| --- |
| select  m.movie,  tbl.cate  from  movie\_info m  lateral view  explode(split(category, ",")) tbl as cate; |

### 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：起点，UNBOUNDED PRECEDING 表示从前面的起点， UNBOUNDED FOLLOWING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

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。

2．数据准备：name，orderdate，cost

jack,2017-01-01,10

tony,2017-01-02,15

jack,2017-02-03,23

tony,2017-01-04,29

jack,2017-01-05,46

jack,2017-04-06,42

tony,2017-01-07,50

jack,2017-01-08,55

mart,2017-04-08,62

mart,2017-04-09,68

neil,2017-05-10,12

mart,2017-04-11,75

neil,2017-06-12,80

mart,2017-04-13,94

3．需求

1. 查询在2017年4月份购买过的顾客及总人数
2. 查询顾客的购买明细及月购买总额
3. 上述的场景, 将每个顾客的cost按照日期进行累加
4. 查询每个顾客上次的购买时间
5. 查询前20%时间的订单信息

4．创建本地business.txt，导入数据

[atguigu@hadoop102 datas]$ vi business.txt

5．创建hive表并导入数据

|  |
| --- |
| create table business(  name string,  orderdate string,  cost int  ) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';  load data local inpath "/opt/module/datas/business.txt" into table business; |

6．按需求查询数据

1. 查询在2017年4月份购买过的顾客及总人数

|  |
| --- |
| select name,count(\*) over ()  from business  where substring(orderdate,1,7) = '2017-04'  group by name; |

1. 查询顾客的购买明细及月购买总额

|  |
| --- |
| select name,orderdate,cost,sum(cost) over(partition by month(orderdate)) from  business; |

1. 上述的场景, 将每个顾客的cost按照日期进行累加

|  |
| --- |
| select name,orderdate,cost,  sum(cost) over() as sample1,--所有行相加  sum(cost) over(partition by name) as sample2,--按name分组，组内数据相加  sum(cost) over(partition by name order by orderdate) as sample3,--按name分组，组内数据累加  sum(cost) over(partition by name order by orderdate rows between UNBOUNDED PRECEDING and current row ) as sample4 ,--和sample3一样,由起点到当前行的聚合  sum(cost) over(partition by name order by orderdate rows between 1 PRECEDING and current row) as sample5, --当前行和前面一行做聚合  sum(cost) over(partition by name order by orderdate rows between 1 PRECEDING AND 1 FOLLOWING ) as sample6,--当前行和前边一行及后面一行  sum(cost) over(partition by name order by orderdate rows between current row and UNBOUNDED FOLLOWING ) as sample7 --当前行及后面所有行  from business;  rows必须跟在Order by 子句之后，对排序的结果进行限制，使用固定的行数来限制分区中的数据行数量 |

1. 查看顾客上次的购买时间

|  |
| --- |
| select name,orderdate,cost,  lag(orderdate,1,'1900-01-01') over(partition by name order by orderdate ) as time1, lag(orderdate,2) over (partition by name order by orderdate) as time2  from business; |

1. 查询前20%时间的订单信息

|  |
| --- |
| select \* from (  select name,orderdate,cost, ntile(5) over(order by orderdate) sorted  from business  ) t  where sorted = 1; |

### 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

2．数据准备

表6-7 数据准备

|  |  |  |
| --- | --- | --- |
| name | subject | score |
| 孙悟空 | 语文 | 87 |
| 孙悟空 | 数学 | 95 |
| 孙悟空 | 英语 | 68 |
| 大海 | 语文 | 94 |
| 大海 | 数学 | 56 |
| 大海 | 英语 | 84 |
| 宋宋 | 语文 | 64 |
| 宋宋 | 数学 | 86 |
| 宋宋 | 英语 | 84 |
| 婷婷 | 语文 | 65 |
| 婷婷 | 数学 | 85 |
| 婷婷 | 英语 | 78 |

3．需求

计算每门学科成绩排名。

4．创建本地score.txt，导入数据

[atguigu@hadoop102 datas]$ vi score.txt

5．创建hive表并导入数据

|  |
| --- |
| create table score(  name string,  subject string,  score int)  row format delimited fields terminated by "\t";  load data local inpath '/opt/module/datas/score.txt' into table score; |

6．按需求查询数据

|  |
| --- |
| select name,  subject,  score,  rank() over(partition by subject order by score desc) rp,  dense\_rank() over(partition by subject order by score desc) drp,  row\_number() over(partition by subject order by score desc) rmp  from score;  name subject score rp drp rmp  孙悟空 数学 95 1 1 1  宋宋 数学 86 2 2 2  婷婷 数学 85 3 3 3  大海 数学 56 4 4 4  宋宋 英语 84 1 1 1  大海 英语 84 1 1 2  婷婷 英语 78 3 2 3  孙悟空 英语 68 4 3 4  大海 语文 94 1 1 1  孙悟空 语文 87 2 2 2  婷婷 语文 65 3 3 3  宋宋 语文 64 4 4 4  扩展：求出每门学科前三名的学生？ |

### 日期相关函数

1. current\_date返回当前日期

select current\_date();

2. date\_add, date\_sub 日期的加减

--今天开始90天以后的日期

select date\_add(current\_date(), 90);

--今天开始90天以前的日期

select date\_sub(current\_date(), 90);

3. 两个日期之间的日期差

--今天和1990年6月4日的天数差

SELECT datediff(CURRENT\_DATE(), "1990-06-04");

# Hive实战之谷粒影音

## 需求描述

统计硅谷影音视频网站的常规指标，各种TopN指标：

--统计视频观看数Top10

--统计视频类别热度Top10

## 项目

### 数据结构

1．视频表

表6-13 视频表

|  |  |  |
| --- | --- | --- |
| 字段 | 备注 | 详细描述 |
| video id | 视频唯一id（String） | 11位字符串 |
| uploader | 视频上传者（String） | 上传视频的用户名String |
| age | 视频年龄（int） | 视频在平台上的整数天 |
| category | 视频类别（Array<String>） | 上传视频指定的视频分类 |
| length | 视频长度（Int） | 整形数字标识的视频长度 |
| views | 观看次数（Int） | 视频被浏览的次数 |
| rate | 视频评分（Double） | 满分5分 |
| Ratings | 流量（Int） | 视频的流量，整型数字 |
| conments | 评论数（Int） | 一个视频的整数评论数 |
| related ids | 相关视频id（Array<String>） | 相关视频的id，最多20个 |

2．用户表

表6-14 用户表

|  |  |  |
| --- | --- | --- |
| 字段 | 备注 | 字段类型 |
| uploader | 上传者用户名 | string |
| videos | 上传视频数 | int |
| friends | 朋友数量 | int |

### ETL原始数据

通过观察原始数据形式，可以发现，视频可以有多个所属分类，每个所属分类用&符号分割，且分割的两边有空格字符，同时相关视频也是可以有多个元素，多个相关视频又用“\t”进行分割。为了分析数据时方便对存在多个子元素的数据进行操作，我们首先进行数据重组清洗操作。即：将所有的类别用“&”分割，同时去掉两边空格，多个相关视频id也使用“&”进行分割。

1．ETL之ETLUtil

|  |
| --- |
| public class ETLUtil {  public static String oriString2ETLString(String ori){  StringBuilder etlString = new StringBuilder();  String[] splits = ori.split("\t");  if(splits.length < 9) return null;  splits[3] = splits[3].replace(" ", "");  for(int i = 0; i < splits.length; i++){  if(i < 9){  if(i == splits.length - 1){  etlString.append(splits[i]);  }else{  etlString.append(splits[i] + "\t");  }  }else{  if(i == splits.length - 1){  etlString.append(splits[i]);  }else{  etlString.append(splits[i] + "&");  }  }  }    return etlString.toString();  }  } |

2．ETL之Mapper

|  |
| --- |
| import java.io.IOException;  import org.apache.commons.lang.StringUtils;  import org.apache.hadoop.io.NullWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Mapper;  import com.atguigu.util.ETLUtil;  public class VideoETLMapper extends Mapper<Object, Text, NullWritable, Text>{  Text text = new Text();    @Override  protected void map(Object key, Text value, Context context) throws IOException, InterruptedException {  String etlString = ETLUtil.oriString2ETLString(value.toString());    if(StringUtils.isBlank(etlString)) return;    text.set(etlString);  context.write(NullWritable.get(), text);  }  } |

3．ETL之Runner

|  |
| --- |
| import java.io.IOException;  import org.apache.hadoop.conf.Configuration;  import org.apache.hadoop.fs.FileSystem;  import org.apache.hadoop.fs.Path;  import org.apache.hadoop.io.NullWritable;  import org.apache.hadoop.io.Text;  import org.apache.hadoop.mapreduce.Job;  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  import org.apache.hadoop.util.Tool;  import org.apache.hadoop.util.ToolRunner;  public class VideoETLRunner implements Tool {  private Configuration conf = null;  @Override  public void setConf(Configuration conf) {  this.conf = conf;  }  @Override  public Configuration getConf() {  return this.conf;  }  @Override  public int run(String[] args) throws Exception {  conf = this.getConf();  conf.set("inpath", args[0]);  conf.set("outpath", args[1]);  Job job = Job.getInstance(conf);    job.setJarByClass(VideoETLRunner.class);    job.setMapperClass(VideoETLMapper.class);  job.setMapOutputKeyClass(NullWritable.class);  job.setMapOutputValueClass(Text.class);  job.setNumReduceTasks(0);    this.initJobInputPath(job);  this.initJobOutputPath(job);    return job.waitForCompletion(true) ? 0 : 1;  }  private void initJobOutputPath(Job job) throws IOException {  Configuration conf = job.getConfiguration();  String outPathString = conf.get("outpath");    FileSystem fs = FileSystem.get(conf);    Path outPath = new Path(outPathString);  if(fs.exists(outPath)){  fs.delete(outPath, true);  }    FileOutputFormat.setOutputPath(job, outPath);    }  private void initJobInputPath(Job job) throws IOException {  Configuration conf = job.getConfiguration();  String inPathString = conf.get("inpath");    FileSystem fs = FileSystem.get(conf);    Path inPath = new Path(inPathString);  if(fs.exists(inPath)){  FileInputFormat.addInputPath(job, inPath);  }else{  throw new RuntimeException("HDFS中该文件目录不存在：" + inPathString);  }  }  public static void main(String[] args) {  try {  int resultCode = ToolRunner.run(new VideoETLRunner(), args);  if(resultCode == 0){  System.out.println("Success!");  }else{  System.out.println("Fail!");  }  System.exit(resultCode);  } catch (Exception e) {  e.printStackTrace();  System.exit(1);  }  }  } |

4．执行ETL

|  |
| --- |
| $ bin/yarn jar ~/softwares/jars/gulivideo-0.0.1-SNAPSHOT.jar \  com.atguigu.etl.ETLVideosRunner \  /gulivideo/video/2008/0222 \  /gulivideo/output/video/2008/0222 |

## 准备工作

### 创建表

创建表：gulivideo\_ori，gulivideo\_user\_ori，

创建表：gulivideo\_orc，gulivideo\_user\_orc

gulivideo\_ori：

|  |
| --- |
| create table gulivideo\_ori(  videoId string,  uploader string,  age int,  category array<string>,  length int,  views int,  rate float,  ratings int,  comments int,  relatedId array<string>)  row format delimited  fields terminated by "\t"  collection items terminated by "&"  stored as textfile; |

gulivideo\_user\_ori：

|  |
| --- |
| create table gulivideo\_user\_ori(  uploader string,  videos int,  friends int)  row format delimited  fields terminated by "\t"  stored as textfile; |

然后把原始数据插入到orc表中

gulivideo\_orc：

|  |
| --- |
| create table gulivideo\_orc(  videoId string,  uploader string,  age int,  category array<string>,  length int,  views int,  rate float,  ratings int,  comments int,  relatedId array<string>)  row format delimited fields terminated by "\t"  collection items terminated by "&"  stored as orc; |

gulivideo\_user\_orc：

|  |
| --- |
| create table gulivideo\_user\_orc(  uploader string,  videos int,  friends int)  row format delimited  fields terminated by "\t"  stored as orc; |

### 导入ETL后的数据

gulivideo\_ori：

|  |
| --- |
| load data inpath "/gulivideo/output/video/2008/0222" into table gulivideo\_ori; |

gulivideo\_user\_ori：

|  |
| --- |
| load data inpath "/gulivideo/user/2008/0903" into table gulivideo\_user\_ori; |

### 向ORC表插入数据

gulivideo\_orc：

|  |
| --- |
| insert into table gulivideo\_orc select \* from gulivideo\_ori; |

gulivideo\_user\_orc：

|  |
| --- |
| insert into table gulivideo\_user\_orc select \* from gulivideo\_user\_ori; |

## 业务分析

### 统计视频观看数Top10

思路：使用order by按照views字段做一个全局排序即可，同时我们设置只显示前10条。

最终代码：

|  |
| --- |
| select  videoId,  uploader,  age,  category,  length,  views,  rate,  ratings,  comments  from  gulivideo\_orc  order by  views  desc limit  10; |

### 统计视频类别热度Top10

思路：

1) 即统计每个类别有多少个视频，显示出包含视频最多的前10个类别。

2) 我们需要按照类别group by聚合，然后count组内的videoId个数即可。

3) 因为当前表结构为：一个视频对应一个或多个类别。所以如果要group by类别，需要先将类别进行列转行(展开)，然后再进行count即可。

4) 最后按照热度排序，显示前10条。

最终代码：

|  |
| --- |
| select  category\_name as category,  count(t1.videoId) as hot  from (  select  videoId,  category\_name  from  gulivideo\_orc lateral view explode(category) t\_catetory as category\_name) t1  group by  t1.category\_name  order by  hot  desc limit  10; |