# Hive基本概念

## 1 什么是Hive

**1） hive简介**

**Hive:**用于解决海量结构化日志的数据统计工具，facebook。

Hive是基于Hadoop的一个数据仓库工具（分析工具），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

运行MapReduce程序，生成相应的分析结果,将结果以表的形式给到client

client

用户根据业务需求编写相应的SQL语句

Hive将SQL语言中常用的操作（select，where，group等）用MapReduce写成很多模板

通过Hive框架匹配出相应的MapReduce模板

所有的MapReduce模板封装在Hive中

（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常用于数据分析，对实时性要求不高的场合，spark。

（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。

Hive架构原理



（1）解析器：将SQL字符串转换成抽象语法树AST，解析器会对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：将AST编译生成逻辑执行计划。

（3）优化器：对逻辑执行计划进行优化。

（4）执行器：把逻辑执行计划转换成可以运行的物理计划，也就是断定当前MR的执行是否需要基于上一步MR的结果。对于Hive来说，就是MR/Spark。

**Metastore：hive元数据(表名，表的数据所在目录等)存储，hive中表的数据不在表中，而是在hdfs中，所以表会与hdfs中的数据产生映射关系。**

**默认存在自带的derby数据库中，但我们用mysql。**

Hive的运行机制：

HQL -> Hive ->数据库查元数据->确定某表与hdfs中某文件存在关联->hive分析数据

Hive不是数据库！！！只是操作起来比较像，Hive的定义是数仓分析工具！并且数据库不能处理海量数据。

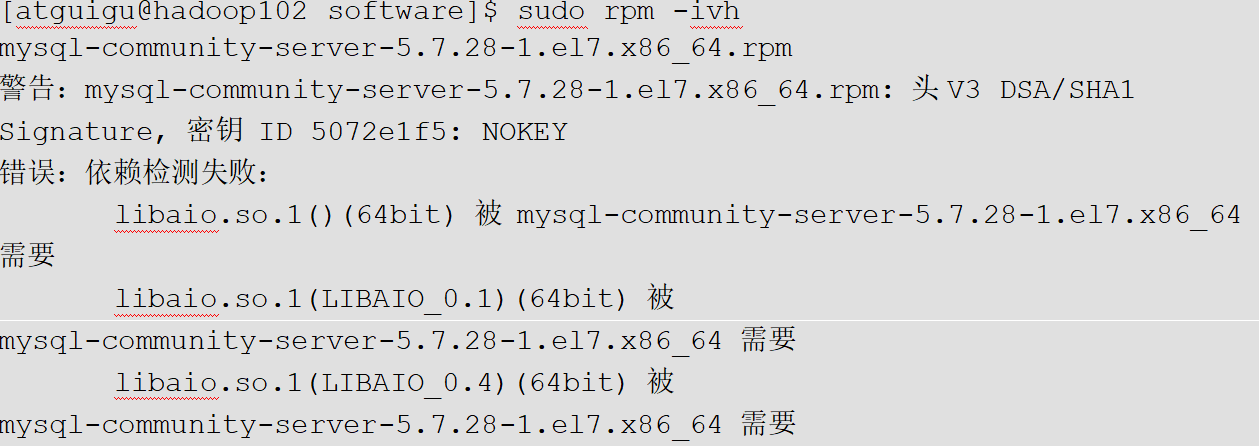
# Hive安装

## 1.MySql安装

为啥需要mysql？原因在于Hive默认使用的元数据库为derby，开启Hive之后就会占用元数据库，且不与其他客户端共享数据，如果想多窗口操作就会报错，操作比较局限。以我们需要将Hive的元数据地址改为MySQL，可支持多窗口操作。

1. **检查当前系统是否安装过Mysql并卸载**
   * + 1. 查询：rpm -qa|grep mariadb（CentOS6->mysql CentOS7 ->mariadb）
       2. **卸载：rpm -qa | grep -E mysql\|mariadb | xargs -n1 sudo rpm -e --nodeps**
2. **安装**
   * + 1. **将MySQL的tar包拷贝到/opt/soft目录下**
       2. **解压MySQL安装包**
       3. **在安装目录下执行rpm安装（按顺序执行）**
          1. **common**
          2. **libs**
          3. **libs-compat**
          4. **client**
          5. **server(安装之前注意安装依赖)**

**安装命令：rpm -ivh rmp包名**



依赖下载：yum install -y libaio

* + - 1. **初始化数据库**

sudo mysqld --initialize --user=mysql

* + - 1. **启动MySQL服务**

sudo systemctl start mysqld

* + - 1. **登录MySQL数据库**

mysql -uroot -p密码

set password = password("新密码")

* + - 1. 解决MySql的登陆问题

mysql -uroot -p密码 -hhadoop101

更改只能本地链接问题

刷新权限：flush privileges;

* + - 1. 涉及中文编码：show variables like “character%”;

## 2.Hive安装

1.上传并解压hive的tar包（嫌名字太长就修改一下名字）

2.配置环境变量

3.解决日志jar包冲突，将hive自带的日志jar包删除。（slf4j）

4.在conf目录中新建一个配置文件，hive-site.xml并添加以下配置：

<?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://hadoop101:3306/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（存放路径） -->

<property>

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

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

</property>

<!-- Hive元数据存储的验证 -->

<property>

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

<value>false</value>

</property>

<!-- 元数据存储授权 -->

<property>

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

<value>false</value>

</property>

</configuration>

由于hive要连接mysql是需要通过jdbc的方式，所以需要jdbc的驱动包，hive通过jdbc的方式连接mysql，所以需要将驱动包放到lib目录下hive才能找得着。

## 完善Hive

初始化元数据库

* 1. **登陆MySQL**
  2. **新建Hive元数据库表空间metastore**
  3. **初始化Hive元数据库**schematool -initSchema -dbType mysql -verbose

## 启动Hive

* 1. 先启动集群
  2. 客户端选择
     1. 直接敲hive启动普通客户端
     2. 启动相对高级的jdbc客户端

<!-- 指定hiveserver2连接的host -->

<property>

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

<value>hadoop101</value>

</property>

<!-- 指定hiveserver2连接的端口号 -->

<property>

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

<value>10000</value>

</property>

* + - 1. cd到bin目录下，其中有个beeline命令，这个家伙就是基于jdbc协议去操作hive的，但hive正常情况下没有提供jdbc协议让咱们去连接，所以非常时期非常操作，hive提供了第二个服务：hiveserver2，它是一个中间商，它能帮助咱们去操作hive。
      2. 启动中间商：hive --service hiveserver2 ,该方式有个缺陷，会停顿，阻塞，而且不能关闭，关闭的话服务就停了。
      3. 通过beeline命令连接中间商（多等一会儿）：

beeline -u jdbc:hive2://hadoop101:10000 -n tedu

### （可选）使用元数据服务的方式访问Hive

**1）在hive-site.xml文件中添加如下配置信息**

<!-- 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>

**2）启动metastore**

[tedu@hadoop101 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

**3）启动 hive**

[tedu@hadoop101 hive]$ bin/hive

脚本相关：

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

0:标准输入

1:标准输出

2:错误输出

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。

nohup hive --service hiveserver2 2>&1 &（不要求掌握）

完整脚本：

#!/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

给脚本添加执行权限。

**启动Hive后台服务**hiveservices start

## 5. Hive常用交互命令

**1.命令hive -help可以查看hive全部的交互命令**

**(1) “-e”不进入hive的交互窗口执行sql语句**

**(2) “-f”执行脚本中sql语句**

**1.创建一个空文件并写入正确命令**

**2.hive -f 文件名**

## 6. Hive的其他命令操作

**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）进入到当前用户的根目录/root或/home/tedu

（2）查看. hivehistory文件

[tedu@hadoop101 ~]$ cat .hivehistory

## Hive常见属性配置

* + 1. hive窗口打印当前库和表头

在hive-site.xml中加入如下两个配置:

<property>

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

<value>true</value>

</property>

<property>

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

<value>true</value>

</property>

* + 1. Hive运行日志信息配置
       1. **Hive的log默认存放在/tmp/tedu/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-log4j2.properties.template文件名称为hive-log4j2.properties
          2. 在hive-log4j.properties文件中修改log存放位置

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

## 参数配置方式

**1）查看当前所有的配置信息**

hive>set;

1. **参数的配置三种方式**

**优先级：由高到低：set关键字 > hiveconf > hive-site.xml 配置文件**

（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：

[tedu@hadoop101 hive]$ bin/hive -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。

## Hive数据类型

## 基本数据类型

|  |  |  |  |
| --- | --- | --- | --- |
| 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

|  |  |  |
| --- | --- | --- |
| 数据类型 | 描述 | 语法示例 |
| STRUCT（对象） | 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。

**1）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": 19 ,

"xiaoxiao song": 18

}

"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 test1(

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 '\_' -- STRUCT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/hive/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)

## 10.类型转换

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。 小的往大的转，大的不能往小的转。

（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。

11：3

2：5

2.0：10

钝角:

**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://hadoop101:10000> select '1'+2, cast('1'as int) + 2;

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

| \_c0 | \_c1 |

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

| 3.0 | 3 |

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

## 11.DDL数据定义

**一.创建数据库：**

基本语法：create database 库名;

可选：

if not exists:如果表不存在则创建，存在则不创建（避免报错产生尴尬）。

Location：在查询该库时可以显示信息。（一般用于指定该库关联hdfs中的目录，不写则默认执行）

with dbproperties:在查询库时显示信息，不会影响库

Comment：描述，记得加引号

1. 查看库
   1. 普通：show databases;
   2. 高级：desc database 库名;
   3. 尼古拉斯赵四:desc database extended 库名;
2. 修改库
   1. alter database 库名 set dbproperties(“....”=”...”,....)
3. 删除库
   1. 普通：drop database 库名;
   2. 强制：drop database 库名 cascade;
   3. 未卜先知: drop database if exists 库名;

**二.创建数据表**

**1.建表语法**

**1).普通**

**CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table\_name -- EXTERANL:**

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

**[ROW FORMAT delimited fields terminated by ... ]**

**[collection items terminated by ... ]**

**[map keys terminated by ... ]**

1. **.大拿**

**CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table\_name -- EXTERANL:**

**[(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, ...)**

**INTO num\_buckets BUCKETS]**

**[ROW FORMAT delimited fields terminated by ... ]**

**[collection items terminated by ... ]**

**[map keys terminated by ... ]**

**[STORED AS file\_format]**

**[LOCATION hdfs\_path]**

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

**[AS select\_statement]**

### 12.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。

**2）案例实操**

（0）原始数据

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

（1）普通创建表

create table if not exists student(

id int, name string

)

row format delimited fields terminated by '\t'

location '/user/hive/warehouse/student';

（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 student2 as select id, name from student;

（3）根据已经存在的表结构创建表

create table if not exists student3 like student;

（4）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）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。

**3）案例实操**

分别创建部门和员工外部表，并向表中导入数据。

（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）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

创建部门表

create external table if not exists dept(

deptno int,

dname string,

loc int

)

row format delimited fields terminated by '\t';

创建员工表

create external table if not exists emp(

empno int,

ename string,

job string,

mgr int,

hiredate string,

sal double,

comm double,

deptno int)

row format delimited fields terminated by '\t';

（3）查看创建的表

hive (default)>show tables;

（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 dept;

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

### 管理表与外部表的互相转换

（1）查询表的类型

hive (default)> desc formatted student2;

Table Type: MANAGED\_TABLE

（2）修改内部表student2为外部表

alter table student2 set tblproperties('EXTERNAL'='TRUE');

（3）查询表的类型

hive (default)> desc formatted student2;

Table Type: EXTERNAL\_TABLE

（4）修改外部表student2为内部表

alter table student2 set tblproperties('EXTERNAL'='FALSE');

（5）查询表的类型

hive (default)> desc formatted student2;

Table Type: MANAGED\_TABLE

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

## 数据导入

### 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 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/hive/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/hive/datas/student.txt /user/atguigu/hive;

加载HDFS上数据

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

加载数据覆盖表中已有的数据

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

### 通过查询语句向表中插入数据（Insert）

**1）创建一张表**

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

**2）基本插入数据**

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

**3）基本模式插入（根据单张表查询结果）**

hive (default)> insert overwrite table student\_par

select id, name from student ;

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

**4）多表（多分区）插入模式（根据多张表查询结果）**

hive (default)> from student

insert overwrite table student partition(month='201707')

select id, name where month='201709'

insert overwrite table student partition(month='201706')

select id, name where month='201709';

### 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;

### 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 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;

### Import数据到指定Hive表中

注意：先用export导出后，再将数据导入。

hive (default)> import table student2 from

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

## 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/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/atguigu/student2'

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

select \* from student;

### Hadoop命令导出到本地

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

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

### Hive Shell 命令导出

基本语法：（hive -f/-e 执行语句或者脚本 > file）

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

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

### Export导出到HDFS上

(defahiveult)> 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。

### Sqoop导出

后续课程专门讲。

### 清除表中数据（Truncate）

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;

# 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]

**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';

1. 创建员工表

create table if not exists emp(

empno int,

ename string,

job string,

mgr int,

hiredate string,

sal double,

comm double,

deptno int)

row format delimited fields terminated by '\t';

1. 导入数据

### 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;

### 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;

### 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

**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。 |

练习：

1. 查询出薪水等于5000的所有员工
2. 查询工资在500到1000的员工信息

（3）查询comm为空的所有员工信息

（4）查询工资是1500或5000的员工信息

**使用LIKE运算选择类似的值**

**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。

练习继续：

1. 查找名字以A开头的员工信息
2. 查找名字中第二个字母为A的员工信息
3. 查找名字中带有A的员工信息

### 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 |

练习再继续：

1. 查询薪水大于1000，部门是30
2. 查询薪水大于1000，或者部门是30
3. 查询除了20部门和30部门以外的员工信息

### 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。

练习再再继续：

1. 计算emp表每个部门的平均工资
2. 计算emp每个部门中每个岗位的最高薪水

### Having语句

**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。

练习再再再继续：

1. 求每个部门的平均薪水大于2000的部门

### 等值Join

Hive支持通常的SQL JOIN语句。

**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;

### 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;

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。

1. 数据准备

1700 Beijing

1800 London

1900 Tokyo

2.创建位置表

create table if not exists location(

loc int,

loc\_name string

)

row format delimited fields terminated by '\t';

3.导入数据

### 笛卡尔积

**1）笛卡尔积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;

### 全局排序（Order By）

Order By：全局排序，只能有一个Reducer

查看reduce个数以及设置：set mapreduce.job.reduces

**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;

### 每个Reduce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;

### 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。

### 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。

# 13.分区表和分桶表

## 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, 通过部门信息模拟）**

dept\_20220307.log

dept\_20220308.log

dept\_20220309.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\_20220307.log

10 ACCOUNTING 1700

20 RESEARCH 1800

dept\_20220308.log

30 SALES 1900

40 OPERATIONS 1700

dept\_20220309.log

50 TEST 2000

60 DEV 1900

1. **加载数据**

hive (default)> load data local inpath '...../dept\_20220307.log' into table dept\_partition partition(day='20220307');

hive (default)> load data local inpath '..../dept\_20220308.log' into table dept\_partition partition(day='20220308');

hive (default)> load data local inpath '..../dept\_20220309.log' into table dept\_partition partition(day='20220309');

注意：分区表加载数据时，必须指定分区

**增加分区**

创建单个分区

hive (default)> alter table dept\_partition add partition(day='20220310') ;

同时创建多个分区

hive (default)> alter table dept\_partition add partition(day='20220311') partition(day='20220312');

**删除分区**

删除单个分区

hive (default)> alter table dept\_partition drop partition (day='20220310');

同时删除多个分区

hive (default)> alter table dept\_partition drop partition (day='20220311'), partition(day='20220312');

**查看分区表有多少分区**

hive> show partitions 表名;

增加分区

删除分区

### 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\_20220310.log' into table

dept\_partition2 partition(day='20220310', hour='12');

（2）查询分区数据

hive (default)> select \* from dept\_partition2 where day='20220310' 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

msck repair table 表名;

（2）方式二：上传数据后添加分区

（3）方式三：创建文件夹后load数据到分区

### 动态分区

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;

思考：目标分区表是如何匹配到分区字段的？

1. **函数**

**窗口函数（开窗）**

**可以将数据容纳到某个空间中进行计算，**

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改变而变化。

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。

**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

**需求**

1. **查询在2017年4月份购买过的顾客及总人数**

需求二：

* + 1. 查询顾客的购买明细及全部人的月购买总额
    2. 查询顾客的购买明细及每个人的月购买总额

1. 上述的场景, 将每个顾客的cost按照日期进行累加(order by)

扩展：求所有顾客的购买明细及按照日期进行排序后

1. 求 所有顾客的cost 第一行 到 当前行 累加
2. 所有顾客的cost 上一行 到 当前行 的累加和
3. 所有顾客的cost 上一行 到 下一行 的累加和
4. 所有顾客的cost 当前行 到 下一行 的累加和
5. 所有顾客的cost 当前行 到 最后一行的累加和
6. 查询每个顾客上次的购买时间
7. 查询前20%时间的订单信息

**创建hive表并导入数据**

create table business(

name string,

orderdate string,

cost int

) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

总结:

over(): 会为每条数据都开启一个窗口. 默认的窗口大小就是当前数据集的大小.

over(partition by ....) : 会按照指定的字段进行分区， 将分区字段的值相同的数据划分到相同的区。

每个区中的每条数据都会开启一个窗口.每条数据的窗口大小默认为当前分区数据集的大小.

over(order by ....) : 会在窗口中按照指定的字段对数据进行排序.

会为每条数据都开启一个窗口,默认的窗口大小为从数据集开始到当前行.

over(partition by .... order by ....) :会按照指定的字段进行分区， 将分区字段的值相同的数据划分到相同的区,在每个区中会按照指定的字段进行排序. 会为每条数据都开启一个窗口,默认的窗口大小为当前分区中从数据集开始到当前行.

over(partition by ... order by ... rows between ... and ....) : 指定每条数据的窗口大小.

### 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**

|  |  |  |
| --- | --- | --- |
| 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。

## 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）

聚集函数，多进一出

类似于：count/max/min

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

一进多出

**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;

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

**5）将jar包添加到hive的classpath**

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

**6）创建临时函数与开发好的java class关联**

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

**7）即可在hql中使用自定义的函数**

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

## 自定义UDTF函数

**0）需求**

自定义一个UDTF实现将一个任意分割符的字符串切割成独立的单词，例如：

hive(default)> select myudtf("hello,world,hadoop,hive", ",");

hello

world

hadoop

hive

**1）代码实现**

package com.atguigu.udtf;

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

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

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

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

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

import org.apache.hadoop.hive.serde2.objectinspector.StructObjectInspector;

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

import java.util.ArrayList;

import java.util.List;

public class MyUDTF extends GenericUDTF {

private ArrayList<String> outList = new ArrayList<>();

@Override

public StructObjectInspector initialize(StructObjectInspector argOIs) throws UDFArgumentException {

//1.定义输出数据的列名和类型

List<String> fieldNames = new ArrayList<>();

List<ObjectInspector> fieldOIs = new ArrayList<>();

//2.添加输出数据的列名和类型

fieldNames.add("lineToWord");

fieldOIs.add(PrimitiveObjectInspectorFactory.javaStringObjectInspector);

return ObjectInspectorFactory.getStandardStructObjectInspector(fieldNames, fieldOIs);

}

@Override

public void process(Object[] args) throws HiveException {

//1.获取原始数据

String arg = args[0].toString();

//2.获取数据传入的第二个参数，此处为分隔符

String splitKey = args[1].toString();

//3.将原始数据按照传入的分隔符进行切分

String[] fields = arg.split(splitKey);

//4.遍历切分后的结果，并写出

for (String field : fields) {

//集合为复用的，首先清空集合

outList.clear();

//将每一个单词添加至集合

outList.add(field);

//将集合内容写出

forward(outList);

}

}

@Override

public void close() throws HiveException {

}

}

**2）打成jar包上传到服务器/opt/module/hive/data/myudtf.jar**

**3）将jar包添加到hive的classpath下**

hive (default)> add jar /opt/module/hive/data/myudtf.jar;

**4）创建临时函数与开发好的java class关联**

hive (default)> create temporary function myudtf as "com.atguigu.hive.MyUDTF";

**5）使用自定义的函数**

hive (default)> select myudtf("hello,world,hadoop,hive",",") ;