第6章Hive数据仓库

(源自:https://biglab.site)

(版本:Ver1.0-20231109)

任务前言

学习目标

1. 了解Hive的概念。
2. 了解Hive与传统数据库的异同点。
3. 了解系统架构及数据模型
4. 熟悉Hive的安装的3种模式及配置过程
5. 掌握Hive中数据库与表的创建、修改操作方法
6. 掌握Hive表数据增删查改的操作方法

任务背景

随着信息技术的普及和企业信息化建设步伐的加快，企业逐步认识到建立企业范围内的统一数据存储的重要性，越来越多的企业已经建立或正着手建立企业数据仓库。企业数据仓库（Data Warehouse）有效集成了来自不同部门、不同地理位置、具有不同格式的数据，为企业管理决策提供了企业范围内的单一数据视图，从而为综合分析和科学决策奠定了坚实的基础。

电信运营商为了优化维护工作，每天都会派出维护人员对分布在全国各的基站进行检测，分析基站的工作情况。其中掉话率在移动通信网中是一项非常重要的指标，掉话率的高低在一定程度上体现了移动网通信质量的优劣。不同厂家的设备对该指标的计算方法与标准不尽相同。因此，对基站掉话率的分析显得尤为重要。

运营商有一个系统，收集了分布在各地基站的各种数据，数据字段说明如下表所示，共有5个数据字段

属性名称	属性说明
record_time	通话时间（单位：s）
imei	基站编号
cell	手机编号
drop_num	掉话秒数（单位：s）
duration	通话总秒数（单位：s）

为了提升网络质量，节约时间和人力成本，优化维护工作，分析基站的掉话率就显得尤为重要，因此，针对系统收集的各基站运行数据，运营商为更精准、有针对性的完成基站的维护工作，提出了基站掉话率统计需求

通常，运营商基站运行期间产生的文件数量都比较大，而且文件中的记录数也很多，文件大多数为文本格式。为了保证较高的处理效率与灵活性，在此选用Hive数据仓库对基站数据进行存储与处理分析。本章将详细讲解怎样通过Hive编程解决实际问题，首先介绍Hive数据仓库的系统架构、数据模式和工作原理，其次讲解访问Hive的3种方式及其搭建过程，再介绍Hive数据仓库中表的创建与修改的基本语法格式，并结合官方的示例介绍表数据的增删改查，最后通过编写Hive语句实现基站掉话率的统计分析

软件准备

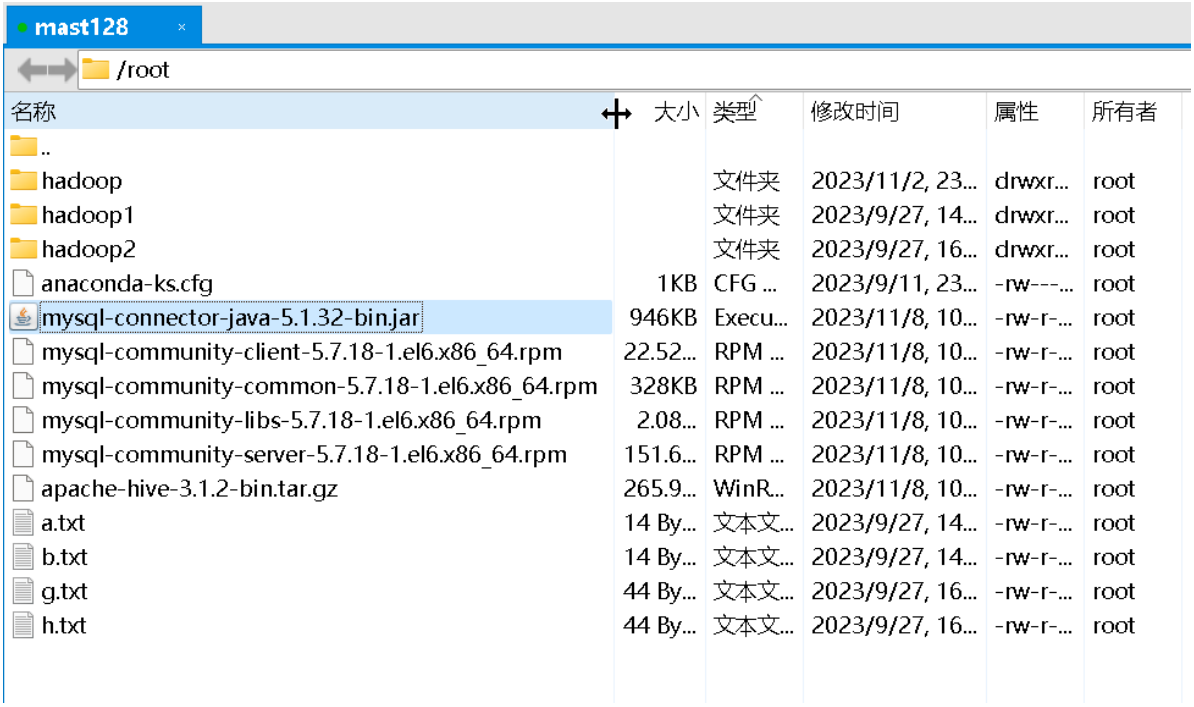
下载地址：链接: https://pan.baidu.com/s/1DHGRrnjvi4yMY41Se7_vDA 提取码: vbgu

软件列表：

软件	版本	安装包称
apache-hive	3.1.2	apache-hive-3.1.2-bin.tar.gz
mysql-connector-java	5.1.32	mysql-connector-java-5.1.32-bin
mysql-community-client	5.7.18	mysql-community-client-5.7.18-1.el6.x86_64.rpm
mysql-community-common	5.7.18	mysql-community-common-5.7.18-1.el6.x86_64.rpm
mysql-community-libs	5.7.18	mysql-community-libs-5.7.18-1.el6.x86_64.rpm
mysql-community-server	5.7.18	mysql-community-server-5.7.18-1.el6.x86_64.rpm

软件上传：

1. 使用xftp 将apache-hive-3.1.2-bin.tar.gz和mysql-connector-java-5.1.32-bin 上传到master虚拟机的/root目录下
2. 使用xftp 将mysql-community相关的文件上传到master虚拟机的/root目录下



测试数据

通过百度网盘下载：

下载地址：链接: https://pan.baidu.com/s/1DHGRrnjvi4yMY41Se7_vDA 提取码: vbgu

软件	大小	版本	安装包称
hadoop_data	99.7M	20230925	hadoop_data.tar.gz

下载hadoop_data.tar.gz后，通过xftp工具上传到master主机的/root/hadoop目录下，然后xshell进入master执行：

```
cd /root/hadoop
tar -zxvf ./hadoop_data.tar.gz
hdfs dfs -mkdir /user/myname/stu
hdfs dfs -put ./hadoop_data/student_info.txt  /user/myname/stu
hdfs dfs -put ./hadoop_data/students.txt  /user/myname/stu
hdfs dfs -put ./hadoop_data/course.txt  /user/myname/stu
hdfs dfs -put ./hadoop_data/sc.txt  /user/myname/stu
hdfs dfs -put ./hadoop_data/jizhan_information.csv  /user/myname/stu
```

任务6.1认识Hive数据仓库

任务描述

Hive数据仓库基于Hadoop开发，是Hadoop生态圈组件之一，是海量数据存储和离线批量处理的常用工具。

本小节的任务如下，这是学习与使用Hive进行数据存储与处理的第一步。

1. 了解Hive的概念。
2. 了解Hive与传统数据库的异同点。
3. 了解系统架构及数据模型。

什么是Hive

随着大数据时代的全面到来，传统数据仓库面临的挑战主要包括以下几个方面。

1. 无法满足快速增长的海量数据存储需求。
2. 无法有效处理不同类型的数据。
3. 计算和处理能力不足。

大多数数据仓库应用程序均是使用关系型数据库进行实现的，并使用SQL作为查询语言。为了提高数据仓库的性能，选择将基于传统关系型数据库的数据仓库构建在Hadoop上，是一个很好的解决思路。对于大量的SQL用户，包括专业数据库设计师和管理员，也包括使用SQL语句从数据仓库中抽取信息的临时用户，急需一个可以提供有效地、合理地组织和使用海量数据的模型，Hive应运而生。

Hive是Hadoop生态系统中必不可少的一个工具，它提供了一种类SQL语言，可以查询存储在Hadoop分布式文件系统（Hadoop Distributed File System，HDFS）中的数据，也可以查询其他与Hadoop集成的文件系统的数据，如MapReduce-FS、Amazon S3，也可以查询数据库中的数据，如HBase和Cassandra.

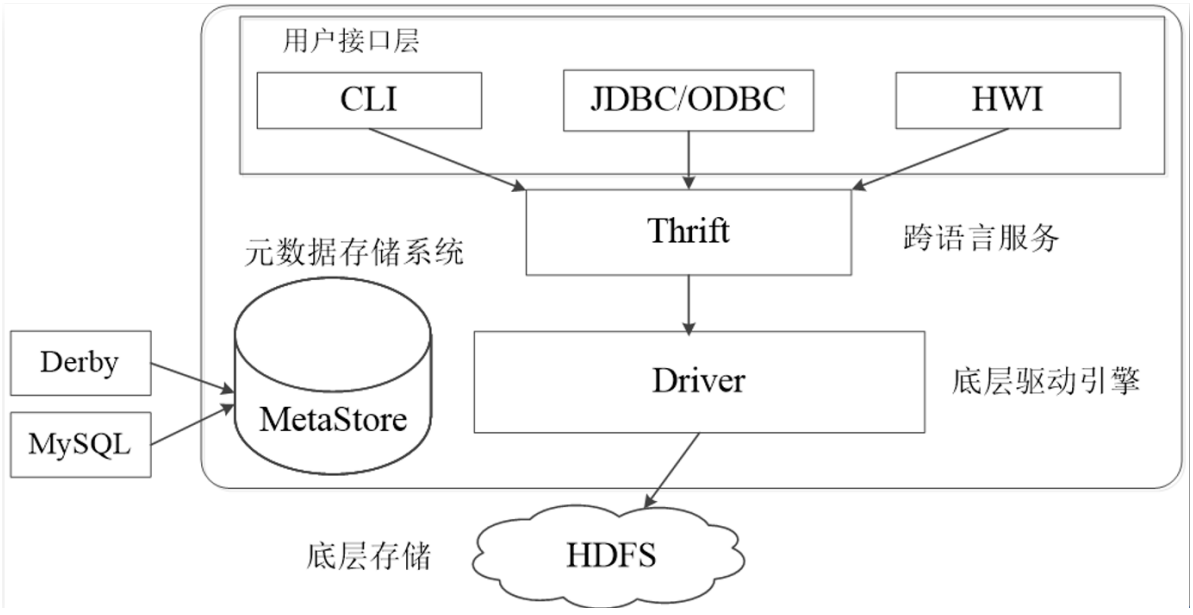
了解Hive与传统数据库的对比

Hive在很多方面和传统数据库类似，但是它底层依赖的是HDFS和MapReduce（或Tez、Spark），因此，Hive在很多方面又有别于传统数据库。从数据存储、索引、分区、执行引擎、执行延迟、扩展性、数据规模等方面，对Hive和传统数据库进行了对比分析，如下表所示

对比内容	Hive	传统数据库
查询语言	HQL	SQL
数据存储	HDFS	本地文件系统
索引	0.8版以后支持位图索引	支持复杂索引
执行引擎	MapReduce、Tez、Spark	自身的执行引擎
执行延迟	高	低
可扩展性	好	有限
处理数据规模	大	小

了解Hive系统架构

Hive是典型的客户端/服务器（Client/Server，C/S）模式，底层执行引擎使用的是Hadoop的MapReduce框架，因此Hive运行在Hadoop基础上。Hive的系统架构组成如下图所示，主要包括5个部分：用户接口层、跨语言服务、元数据存储系统、底层的驱动引擎、底层存储



Hive系统架构说明：

- 用户接口层：

用户接口用于连接访问Hive，包括CLI、JDBC/ODBC和HWI（Hive Web Interface）3种方式。CLI指的是shell终端命令行，采用交互式的方式访问Hive。JDBC/ODBC是通过客户端的方式连接并访问Hive。HWI则是通过浏览器的方式访问Hive。

- 跨语言服务

Thrift是用于进行可扩展且跨语言的服务的开发，Hive集成了Thrift服务，能让不同的编程语言调用Hive的接口

- 元数据库（MetaStore）存储系统

Hive将元数据存储在数据库中，如MySQL、Derby（Hive自带的内存数据库）。Hive中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等。MetaStore默认存在自带的Derby数据库中。由于Derby数据库不适合多用户操作，并且数据存储目录不固定，不方便管理，因此，通常都将元数据存储在MySQL数据库中。

- 底层的驱动引擎（Driver）

底层的驱动引擎实现了将HiveQL语句转化为MapReduce任务的过程。Hive的底层的驱动引擎主要包括解释器、编译器、优化器和执行器，一同完成HQL查询语句从词法分析、语法分析、编译、优化以及查询计划的生成。生成的查询计划存储在HDFS中，并在随后由执行器（Executor）调用MapReduce执行。

- 底层存储

Hive数据仓库的数据是存储在HDFS中的。针对大部分的HQL查询请求，Hive内部会将HQL语句自动转换为MapReduce任务执行。

了解Hive数据模型

- 数据库

Hive数据库类似于传统数据库的Database。内部表与数据库中的table在概念上类似。每一个table在Hive中都有一个相应的目录存储数据，所有的table数据均保存在这个目录中。

- 表

Hive的表在逻辑上由存储的数据和描述表格数据形式的相关元数据组成。表存储的数据存放在分布式文件系统里，如HDFS。

Hive中的表分为两种类型，一种为内部表，这种表的数据存储在Hive数据仓库中；另一种为外部表，表数据可以存放在Hive数据仓库外的分布式文件系统中，也可以存储在Hive数据仓库中。值得一提的是，Hive数据仓库其实也是HDFS中的一个目录，该目录是Hive数据存储的默认路径，也可以在Hive的配置文件中重新配置。

- 分区和桶

Hive将表组织成分区（partition），这是一种根据分区列（partition column）的值对表进行粗略划分的机制。使用分区可以加快数据分片的查询速度。表或分区可以进一步分为桶（bucket）。

将表或分区组织成桶有以下两个原因。

第一个原因是可以获得更高的查询处理效率。桶其实是为表加上了额外的结构，Hive在处理有些查询时能够利用这个结构。如连接两个在相同列上划分了桶的表，可以使用Map端的join高效实现。

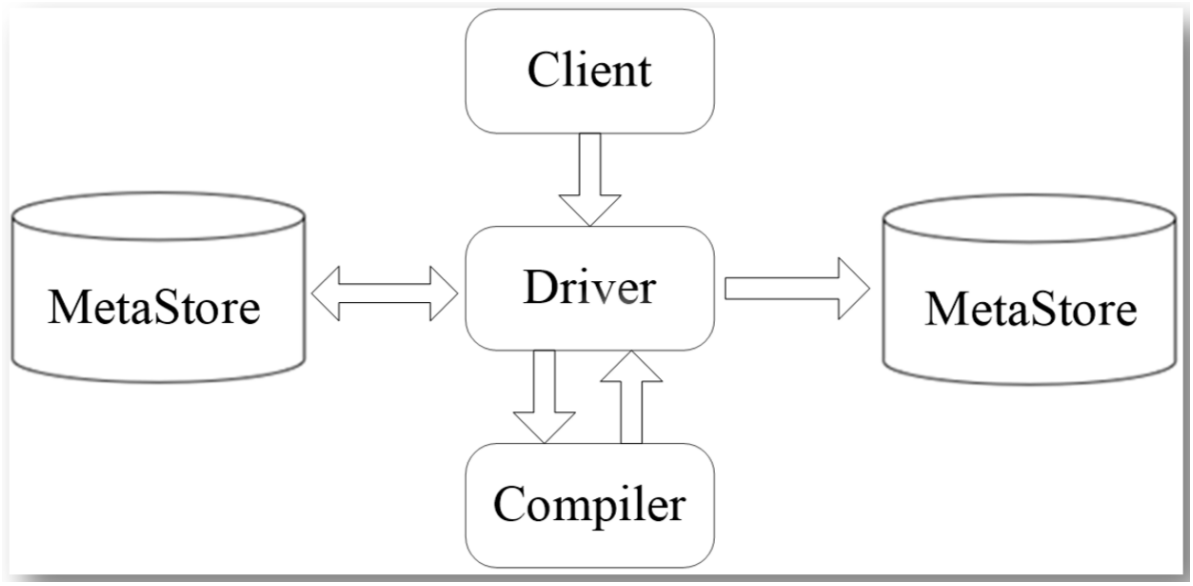
第二个原因是使取样（sampling）更高效。在大规模数据集上进行开发和查询时，如果可以先在一部分小数据集上运行查询，那么会更加地方便。Hive使用CLUSTERED BY子句来指定划分桶所用的列和要划分的桶的个数。

- 视图

Hive的视图与传统数据库的视图类似。视图是只读的，视图依赖的基本表如果改变，数据增加不会影响视图的呈现；如果删除基本表，那么会出现问题。

了解Hive工作原理

Hive的执行流程如下图所示，Hive中的任务执行流程是将客户端（Client）提交的任务（如HiveQL语句）通过驱动（Driver）找到HiveQL中的相关元数据（MetaStore）确定表及存储的情况，再编译（Compiler）相关信息，生成执行计划，提交给Hadoop完成Hive的执行过程



任务6.2访问Hive的3种方式

任务描述

Hive的安装模式分为3种，分别是内嵌模式、直连数据库模式（或本地模式）和远程模式。

1. 内嵌模式使用内嵌的Derby数据库存储元数据。
2. 直连接数据库模式采用外部数据库存储元数据。
3. 远程模式也采用外部数据库存储元数据，不同的是，远程模式需要单独开启MetaStore服务。

为了顺利地访问Hive，本小节的任务是介绍访问Hive的3种方式，并根据不同的安装模式进行配置

设置内嵌Derby模式

内嵌模式使用内嵌的Derby数据库存储元数据。

在内嵌模式下，只允许一个会话连接，若尝试多个会话则连接时将报错。

在master主机上执行安装：

```
cd /root
tar -zxvf ./apache-hive-3.1.2-bin.tar.gz
mv ./apache-hive-3.1.2-bin /usr/local/hive
mv /usr/local/hive/lib/guava-19.0.jar /usr/local/hive/lib/guava-19.0.jar.bak
cp /usr/local/hadoop-3.1.4/share/hadoop/common/lib/guava-27.0-jre.jar
/usr/local/hive/lib/guava-27.0.jar
mv /usr/local/hive/lib/log4j-slf4j-impl-2.10.0.jar /usr/local/hive/lib/log4j-
slf4j-impl-2.10.0.jar.bak
cd /usr/local/hive/bin
./schematool -dbType derby -initSchema
```

运行后会出现如下内容，说明元数据库初始化完成

```
Initialization script completed
schemaTool completed
```

修改系统环境变量

```
vi /etc/profile
```

在文件末尾添加：

```
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin
```

立即生效环境变量

```
source /etc/profile
```

运行hive

(前提是hadoop集群已启动，使用jps查看，若没有，使用start-all.sh启动hadoop集群)

```
hive
```

运行结果如：

```
[root@master bin]# hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-3.1.4/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop-3.1.4/share/hadoop/common/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
which: no hbase in (/usr/local/hadoop-3.1.4/sbin:/usr/local/hadoop-3.1.4/bin:/usr/local/hadoop-3.1.4/sbin:/usr/local/hadoop-3.1.4/bin:/usr/local/hadoop-3.1.4/sbin:/usr/local/hadoop-3.1.4/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/java/jdk1.8.0_281-amd64/bin:/root/bin:/usr/java/jdk1.8.0_281-amd64/bin:/bin:/usr/local/hive/bin:/usr/java/jdk1.8.0_281-amd64/bin:/usr/local/hive/bin)
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-3.1.4/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop-3.1.4/share/hadoop/common/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
```

```
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Hive Session ID = ef4481eb-f38c-4f15-8d19-2028699da4c1

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Hive Session ID = a55a7daa-edef-4ce4-9358-a65a1aa457b1
hive>
```

查看hive:

```
hive> show databases;
OK
default
Time taken: 0.483 seconds, Fetched: 1 row(s)
hive>
hive> exit;
[root@master bin]#
```

说明数据库已正常。

退出hive命令行，使用exit;

设置直连数据库模式

直连数据库模式也叫本地模式，它和远程模式安装配置方式大致相同，本质上是将Hive默认的元数据存储介质由自带的Derby数据库替换为MySQL数据库。无论在什么目录下以任何方式启动Hive，只要连接的是同一台Hive服务，所有节点访问的元数据信息均是一致的，从而实现元数据的共享。以直连数据库模式为例，讲解Hive的安装过程。直连数据库模式的Hive安装主要包括安装MySQL服务和安装配置Hive两个步骤.

以下在master主机上执行:

安装Mysql和配置mysql库

删除已有mysql安装新mysql

```
rm -rf /usr/lib/mysql
rm -rf /usr/include/mysql
rm -rf /etc/my.cnf
rm -rf /var/lib/mysql
rm -rf /usr/share/mysql
cd /root
yum remove -y mariadb-libs
yum install -y net-tools
yum install -y perl
yum remove mysql mysql-server
yum install -y mariadb-libs
rpm -ivh mysql-community-* --force --nodeps
cp /root/mysql-connector-java-5.1.32-bin.jar  /usr/local/hive/lib/
```

修改mysql配置

```
vi /etc/my.cnf
```

添加如下内容：

```
character_set_server=utf8
[client]
default-character-set=utf8
[mysql]
default-character-set=utf8
```

my.cnf文件结果如：

```
# For advice on how to change settings please see
# http://dev.mysql.com/doc/refman/5.7/en/server-configuration-defaults.html

[mysqld]
#
# Remove leading # and set to the amount of RAM for the most important data
# cache in MySQL. Start at 70% of total RAM for dedicated server, else 10%.
# innodb_buffer_pool_size = 128M
#
# Remove leading # to turn on a very important data integrity option: logging
# changes to the binary log between backups.
# log_bin
#
# Remove leading # to set options mainly useful for reporting servers.
# The server defaults are faster for transactions and fast SELECTs.
# Adjust sizes as needed, experiment to find the optimal values.
# join_buffer_size = 128M
# sort_buffer_size = 2M
# read_rnd_buffer_size = 2M
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock

# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0

log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid

character_set_server=utf8
[client]
default-character-set=utf8
[mysql[
default-character-set=utf8
~
~
~
```

启mysql服务

```
systemctl start mysqld
systemctl enable mysqld
service mysqld restart
```

查看临时密码

```
more /var/log/mysqld.log  |grep password |grep generated
```

结果如下，其中: "K6UdV+XTmoAB"，为本机临时密码，每台机器密码都不同

```
2023-11-08T03:46:49.898572Z 1 [Note] A temporary password is generated for root@localhost: K6UdV+XTmoAB
[root@master ~]
```

使用临时密码登陆:

```
mysql -uroot -p'K6UdV+XTmoAB'
```

-p后的临时密码需要替换成本机查询出来的密码。

修改mysql的root密码

进入mysql命令后，修改mysql的root密码为:123456

```
alter user 'root'@'localhost' identified by '@Root_123456';
set global validate_password_policy=0;
set global validate_password_length=6;
set password for 'root'@'localhost'=password('123456');
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '123456' WITH GRANT OPTION;
FLUSH PRIVILEGES;
create database hive;
create database hive_remote;
exit;
```

运行结果如:

```
mysql> alter user 'root'@'localhost' identified by '@Root_123456';
Query OK, 0 rows affected (0.00 sec)

mysql> set global validate_password_policy=0;
Query OK, 0 rows affected (0.00 sec)

mysql> set global validate_password_length=6;
Query OK, 0 rows affected (0.00 sec)

mysql> set password for 'root'@'localhost'=password('123456');
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '123456' WITH GRANT OPTION;
```

```
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql>  exit;
```

配置Hive直连数据库

仍在master配置，所以安装hive过程，/etc/profile环境变量配置不变。

需要配置hive-site.xml

```
cd /usr/local/hive/conf/
vi ./hive-site.xml
```

hive-site.xml内容设置如下：

```
<configuration>
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://master:3306/hive?
createDatabaseIfNotExist=true&amp;useSSL=false&amp;useUnicode=true&amp;chara
cterEncoding=UTF-8</value>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.jdbc.Driver</value>
<description>Driver class name for a JDBC metastore</description>
</property>
<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>root</value>
<description>Username to use against metastore database</description>
</property>
<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>123456</value>
<description>password to use against metastore database</description>
</property>
<property>
<name>hive.metastore.event.db.notification.api.auth</name>
<value>false</value>
</property>
<property>
<name>hive.metastore.schema.verification</name>
<value>false</value>
</property>
</configuration>
```

（！请注意，第4行和5行之前是没有换行的，同学们需要在记事本同合并一行）

初始化元数据库

```
cd /usr/local/hive/bin
./schematool -dbType mysql -initSchema
```

运行后会出现如下completed的内容，说明元数据库初始化完成

```
Initialization script completed
schemaTool completed
[root@master bin]#
```

运行hive

(前提是hadoop集群已启动，使用jps查看，若没有，使用start-all.sh启动hadoop集群)

```
hive
```

运行结果如:

```
[root@master bin]# hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-3.1.4/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop-3.1.4/share/hadoop/common/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
which: no hbase in (/usr/local/hadoop-3.1.4/sbin:/usr/local/hadoop-3.1.4/bin:/usr/local/hadoop-3.1.4/sbin:/usr/local/hadoop-3.1.4/bin:/usr/local/hadoop-3.1.4/sbin:/usr/local/hadoop-3.1.4/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/java/jdk1.8.0_281-amd64/bin:/root/bin:/usr/java/jdk1.8.0_281-amd64/bin:/bin:/usr/local/hive/bin:/usr/java/jdk1.8.0_281-amd64/bin:/usr/local/hive/bin)
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/hadoop-3.1.4/share/hadoop/common/lib/slf4j-log4j12-1.7.25.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/hadoop-3.1.4/share/hadoop/common/slf4j-log4j12-1.7.30.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Hive Session ID = 12077e17-f378-4958-8790-dba9205eec27

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

```
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
Hive Session ID = 7bcad5dd-93b8-473b-a126-ed08898be1fa
hive>
hive> show databases;
OK
default
Time taken: 0.851 seconds, Fetched: 1 row(s)
hive> exit;
```

至此，完成了Hive直连数据库的模式。

设置远程模式

修改master上的配置文件hive-site.xml

```
cd /usr/local/hive/conf/
cp ./hive-site.xml ./hive-site.xml.direct
vi ./hive-site.xml
```

hive-site.xml修改内容如下：

其中有修改行8，**将hive库改为hive_remote:**

jdbc:mysql://master:3306/**hive_remote**?createDatabaseIfNotExist=true&useSSL=false&useUnicode=true&characterEncoding=UTF-8

```
<configuration>
<property>
<name>hive.metastore.warehouse.dir</name>
<value>/user/hive_remote/warehouse</value>
</property>
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:mysql://master:3306/hive_remote?createDatabaseIfNotExist=true&amp;useSSL=false&amp;useUnicode=true&amp;characterEncoding=UTF-8</value>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>com.mysql.jdbc.Driver</value>
<description>Driver class name for a JDBC metastore</description>
</property>
<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>root</value>
<description>Username to use against metastore database</description>
</property>
<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>123456</value>
<description>password to use against metastore database</description>
</property>
<property>
```

```
<name>datanucleus.schema.autoCreateAll</name>
<value>true</value>
</property>
<property>
<name>hive.metastore.schema.verification</name>
<value>false</value>
</property>
</configuration>
```

再次初始化元数据库

```
cd /usr/local/hive/bin
./schematool -dbType mysql -initSchema
```

运行后会出现如下completed的内容，说明元数据库初始化完成

```
Initialization script completed
schemaTool completed
[root@master bin]#
```

从master分发hive到从机slave1,slave2

```
scp -r /usr/local/hive  slave1:/usr/local/
scp -r /usr/local/hive  slave2:/usr/local/
scp /etc/profile slave1:/etc/
scp /etc/profile slave2:/etc/
```

在从机slave1和 slave2上执行

```
source /etc/profile
```

在从机slave1和slave2上修改hive-site.xml文件

```
cd /usr/local/hive/conf/
vi ./hive-site.xml
```

hive-site.xml内容如下：

```
<configuration>
<property>
<name>hive.metastore.warehouse.dir</name>
<value>/user/hive_remote/warehouse</value>
</property>
<property>
<name>hive.metastore.uris</name>
<value>thrift://master:9083</value>
</property>
</configuration>
```

在master上启动hive服务

```
hive --service metastore &
```

(&表示后台运行，不加&将在前台运行，这时不能进行其它操作，Ctrl+C 将退出hive服务)

创建shell文件执行

```
vi /root/start-hive.sh
chmod 777 /root/start-hive.sh
```

start-hive.sh的内容如：

```
hive --service metastore
```

后面启动master的hive服务就可以使用

```
cd /root
./start-hive.sh
```

在slave1或slave2上运行hive

```
hive
```

结果如：

```
[root@slave2 local]# hive
which: no hbase in (/usr/local/hadoop-3.1.4/sbin:/usr/local/hadoop-3.1.4/bin:/usr/local/hadoop-3.1.4/sbin:/usr/local/hadoop-3.1.4/bin:/usr/local/hadoop-3.1.4/sbin:/usr/local/hadoop-3.1.4/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/usr/java/jdk1.8.0_281-amd64/bin:/root/bin:/usr/java/jdk1.8.0_281-amd64/bin:/usr/java/jdk1.8.0_281-amd64/bin:/usr/local/hive/bin)
Hive Session ID = 8f86b1cb-026f-4105-9ced-086bb53bd27a

Logging initialized using configuration in jar:file:/usr/local/hive/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.Hive Session ID = 5c2f0684-fd26-4ff4-b78b-5b7e8c1e4641

hive> show databases;
OK
default
Time taken: 0.618 seconds, Fetched: 1 row(s)
hive>
```

在slave1或slave2创建hive_remote数据库

```
create database hive_remote;
```

运行结果：

```
hive> create database hive_remote;
OK
Time taken: 1.196 seconds
hive> show databases;
OK
default
hive_remote
Time taken: 0.043 seconds, Fetched: 2 row(s)
hive>
```

在master 的mysql库观察

有新增hive_remote数据库

```
mysql -uroot -p123456
```

如

```
[root@master ~]# mysql -uroot -p123456
mysql: [Warning] Using a password on the command line interface can be
insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 275
Server version: 5.7.18 MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

mysql>
```

查看数据库

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| hive               |
| hive_remote        |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
6 rows in set (0.00 sec)
```

修改数据库的字符集

为后续的创建hive表准备

```
ALTER DATABASE hive CHARACTER SET latin1;
ALTER DATABASE hive_remote CHARACTER SET latin1;
```

任务6.3实现Hive表的创建与修改

尽管HiveQL的语法规则与SQL语法非常相似，但是，Hive在模式设计上与传统关系型数据库非常不同。Hive是反模式的，且在反模式的应用中，与传统数据库也是有区别的。

本小节的任务如下。

1. 了解Hive的基本语法。
2. 掌握Hive表的创建与修改。

Hive数据定义语言DDL的基本语法

Hive数据定义语言（DDL）可以实现Hive数据存储所依赖的数据库及表结构的定义、表描述查看和表结构更改的操作。与传统数据库（如MySQL）用法类似，Hive需要先创建数据库，再创建表，表中有表结构，数据按表结构进行存储。与传统数据库相比，Hive对表格式的定义更加宽松、随性。

数据库的基本操作

```
使用CREATE命令可以创建数据库，创建数据库语法格式如下。
CREATE (DATABASE | SCHEMA) [IF NOT EXISTS] database_name
  [COMNENT database_comment]
  [LOCATION hdfs_path]
  [ WITHDBPROPERTIES (property_name=property_value,...)];

使用DROP命令可以删除数据库，删除数据库语法格式如下。
DROP(DATABASE | SCHEMA) [IF EXISTS] database_name [RESTRICTI | CASCADE];

使用ALTER命令可以更改数据库当前目录，更改数据库语法格式如下。
ALTER (DATABASE | SCHEMA) database_name SET LOCATION hdfs_path;
--(Note: Hive2.2.1,2.4.and later)

使用数据库语法格式如下。
USE database_name;
```

```
USE DEPAULT;

```

表的基本操作

```
其中Hive中创建表的基本语法格式如下。
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]

Hive中复制表的基本语法格式如下。
CREATE [TEMPRORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
LIKE existing_table_or_view_name [LOCATION hdfs_path]

Hive中删除表的基本语法格式如下。
DROP TABLE [IF EXISTS] table_name [PURGE]
-- (Note: PURGE available in Hive 0.14.0 and later)

通过RENAME语句可以将表的名称更改为其他名称，语法格式如下。
ALTER TABLE table_name RENAME TO new_table_name;

通过ALTER TABLE语句可以添加或删除表约束，语法格式如下。
ALTER TABLE table_name ADD CONSTRAINT constraint_name PRIMARY KEY (column,
...) DISABLE NOVALIDATE;
ALTER TABLE table_name ADD CONSTRAINT constraint_name FOREIGN KEY (column,
...) REFERENCES table_name(column, ...) DISABLE NOVALIDATE RELY;
ALTER TABLE table_name CHANGE COLUMN column_name column_name data_type
CONSTRAINT constraint_name NOT NULL ENABLE;
ALTER TABLE table_name DROP CONSTRAINT constraint_name;

通过ALTER TABLE语句也可以添加或删除表的分区，语法格式如下。
ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec
[LOCATION 'location'][, PARTITION partition_spec [LOCATION 'location'],
...];
partition_spec:
(partition_column = partition_col_value, partition_column =
partition_col_value, ...)
ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec,
partition_spec, ...]
[IGNORE PROTECTION] [PURGE];


```

Hive表的基本操作

准备测试数据，如果前面已完成数据准备，这里上传指令可以忽略，在master 或slave上执行都可以：

```
cd /root/hadoop
tar -zxvf ./hadoop_data.tar.gz
hdfs dfs -mkdir /user/myname/stu
hdfs dfs -put ./hadoop_data/student_info.txt  /user/myname/stu
```

修改mysql字符集，在master主机上修改； 不修改的话在创建hive表会报错：Column length too big for column 'TYPE_NAME'

```
ALTER DATABASE hive CHARACTER SET latin1;
ALTER DATABASE hive_remote CHARACTER SET latin1;
```

运行结果如：

```
mysql> ALTER DATABASE hive CHARACTER SET latin1;
Query OK, 1 row affected (0.00 sec)

mysql> ALTER DATABASE hive_remote CHARACTER SET latin1;
Query OK, 1 row affected (0.00 sec)

mysql> exit;
```

表的基本操作：内部表，外部表，分区表； 在slave1上的hive命令行执行：

```
-- 创建内部表
create database student;
use student;
create table user_info(id int,name string);
-- 创建外部表
create external table student_info(
stu_no string,
stu_name string,
stu_sex string,
telephone string,
stu_class string)
row format delimited fields terminated by ',' location '/user/myname/stu';
show tables;
select * from user_info;
select * from student_info;
desc student_info;
-- 创建分区表
create table teacher_info(t_no string,t_name string,t_sex string,t_age
string)
partitioned by (deppart string)
row format delimited fields terminated by ',' stored as textfile;
-- 创建表score
create table score(
    stu_no string,
    cla_no string ,
    grade float)
```

```
partitioned by (class_name string);
-- 修改表
alter table score rename to stu_score;
-- 修改表，添加字段
alter table stu_score add columns (credit int, gpa float);
-- 修改表，修改字段
alter table stu_score change column credit Credits float;
-- 修改表，新增分区
alter table stu_score add partition(class_name ='20231109');
-- 修改表，删除分区
alter table stu_score drop if exists partition(class = '20231109');
-- 删除表
drop table stu_score;
```

运行结果如：

```
hive>  create table user_info (id int ,name string);
OK
Time taken: 0.642 seconds
hive> -- 创建外部表
hive> create external table student_info(
    > stu_no string,
    > stu_name string,
    > stu_sex string,
    > telephone string,
    > stu_class string)
    > row format delimited fields terminated by ',' location
'/user/myname/stu';
OK
Time taken: 0.13 seconds
hive>
hive> show tables;
OK
student_info
user_info
Time taken: 0.04 seconds, Fetched: 2 row(s)
hive> select * from user_info;
OK
Time taken: 1.945 seconds
hive> select * from student_info;
OK
stu_no  stu_name        stu_sex telphone        stu_class
2021513501      张晓娟  女      153*****506     2021级信管1班
2021513505      李小鹏  男      156*****501     2021级信管2班
2021513503      张丽    女      186*****556     2021级信管1班
2021513504      刘向东  男      133*****314     2021级信管2班
2021513505      李明博  男      137*****049     2021级信管1班
2021513506      张子强  男      135*****379     2021级信管2班
2021513507      王文海  男      130*****410     2021级信管1班
2021513500      刘慧娟  女      130*****140     2021级信管2班
2021513509      陈敏    女      133*****550     2021级信管1班
2021513510      王云    男      100*****475     2021级信管2班
2021513511      沈璐    女      156*****675     2021级信管1班
2021513515      赵鹏飞  男      133*****514     2021级信管3班
2021513513      谢朝阳  男      153*****557     2021级信管4班
```

```
2021513514      刘子歌  女      135*****533     2021级信管3班
2021513515      陈照升  男      130*****500     2021级信管1班
2021513516      秦小路  女      153*****505     2021级信管3班
2021513517      张大宝  男      133*****541     2021级信管4班
2021513510      刘佳    女      133*****453     2021级信管3班
2021513519      李伟    男      133*****571     2021级信管1班
2021513550      杨荣    女      151*****006     2021级信管3班
2021513551      李大双  男      131*****503     2021级信管1班
2021513555      赵晓雨  女      109*****050     2021级信管5班
2021513553      王大红  女      150*****740     2021级信管4班
2021513554      张雨滴  女      176*****059     2021级信管5班
2021513555      李斯    女      150*****559     2021级信管4班
2021513556      秦小宝  男      153*****056     2021级信管5班
2021513557      魏中祥  男      177*****360     2021级信管4班
2021513550      于媛媛  女      157*****156     2021级信管5班
2021513559      张耗    男      130*****373     2021级信管4班
2021513530      张春晓  女      177*****903     2021级信管5班
2021513531      章小玉  女      153*****459     2021级信管4班
2021513535      林飞飞  女      177*****774     2021级信管5班
2021513533      朱云云  女      173*****779     2021级信管4班
2021513534      夏晓晓  女      135*****550     2021级信管5班
Time taken: 0.177 seconds, Fetched: 35 row(s)
hive>
hive> desc student_info;
OK
stu_no                  string
stu_name                string
stu_sex                 string
telephone               string
stu_class               string
Time taken: 0.057 seconds, Fetched: 5 row(s)
hive>
hive> -- 创建分区表
hive> create table teacher_info(t_no string,t_name string,t_sex
string,t_age string)
    > partitioned by (deppart string)
    > row format delimited fields terminated by ',' stored as textfile;
OK
Time taken: 0.113 seconds
hive>
hive> -- 创建表score
hive> create table score(
    >     stu_no string,
    >     cla_no string ,
    >     grade float)
    > partitioned by (class_name string);
OK
Time taken: 0.11 seconds
hive>
hive> -- 修改表
hive> alter table score rename to stu_score;
OK
Time taken: 0.253 seconds
hive>
hive> -- 修改表，添加字段
```

```
hive> alter table stu_score add columns (credit int, gpa float);
OK
Time taken: 0.141 seconds
hive>
hive> -- 修改表，修改字段
hive> alter table stu_score change column credit Credits float;
OK
Time taken: 0.126 seconds
hive>
hive> -- 修改表，新增分区
hive> alter table stu_score add partition(class_name ='20231109');
OK
Time taken: 0.274 seconds
hive> -- 修改表，删除分区
hive> alter table stu_score drop if exists partition(class = '20231109');
FAILED: NullPointerException null
hive> -- 删除表
hive> drop table stu_score;
OK
Time taken: 0.357 seconds
hive>
```

任务6.4实现Hive表中数据的增删查改

Hive数据操作语言DML的基本语法

```
-- 装载数据。
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename
[PARTITION
 (partcol1=val1, partcol2=val2 ...)]
LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename
[PARTITION
 (partcol1=val1, partcol2=val2 ...)] [INPUTFORMAT 'inputformat' SERDE
'serde']

-- 查询数据格式如下。
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 [offset,] rows]

-- 插入数据格式如下。
INSERT OVERWRITE TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2
...) [IF NOT EXISTS]] select_statement1 FROM from_statement;
INSERT INTO TABLE tablename1 [PARTITION (partcol1=val1, partcol2=val2 ...)]
select_statement1 FROM from_statement;

```

```
-- 删除数据格式如下。
DELETE FROM tablename [WHERE expression];

```

Hive表的基本数据操作

使用LOAD命令将hdfs上的3个文件的数据分别装载至Hive中的3个表中

参考数据准备时已将数据文件导入到hdfs:

```
hdfs dfs -put ./hadoop_data/student_info.txt  /user/myname/stu
hdfs dfs -put ./hadoop_data/students.txt  /user/myname/stu
hdfs dfs -put ./hadoop_data/course.txt  /user/myname/stu
hdfs dfs -put ./hadoop_data/sc.txt  /user/myname/stu
```

以下在 slave1或slave2中执行load命令

```
-- 创建数据表
create table student(Sno int,Sname string,Sex string,Sage int,Sdept string)
row format delimited fields terminated by ',' stored as textfile;
create table course(Cno int,Cname string)
row format delimited fields terminated by ',' stored as textfile;
create table sc(Sno int,Cno int,Grade int)
row format delimited fields terminated by ',' stored as textfile;
-- 向数据表中装载文件
load data inpath '/user/myname/stu/students.txt' overwrite into table
student;
load data inpath '/user/myname/stu/course.txt' overwrite into table course;
load data inpath '/user/myname/stu/sc.txt' overwrite into table sc;
-- 查询数据
select Sno,Sname from student;
select Sno,Sname from student where Sex='男';
select distinct Sname from student inner join sc on student.Sno=sc.Sno;
-- 查询数据，查询学生的课程成绩情况
select student.Sname,Course.Cname,Sc.Grade from student join sc
on student.Sno=sc.Sno join course on sc.cno=course.cno;
-- 查询数据，查询选修2号课程且成绩为90分以上的学生
select student.Sname,sc.Grade from student join sc
on student.Sno=sc.Sno and sc.Grade>90;
-- 查询数据，查询各个课程及相应的选课人数
select Cno,count(1) from sc group by Cno;
-- 查询数据，查询选修了3门以上的课程的学生学号
select Sno from sc group by Sno having count(Cno)>3;
-- 查询数据，查询学生信息并根据学号进行升序排序
select Sno from student order by Sno;
-- 查询数据，按性别分区，在分区内按年龄升序排序
set mapred.reduce.tasks=2;
insert overwrite directory '/user/myname/stu_out'
select * from student
distribute by Sex sort by Sage;
-- 插入数据
-- 插入数据,在student表中插入一条新记录
insert into table student values(2018213223,'王小哲','男',18,'IS');
-- 删除数据

```

```
delete from student where Sno='2018213223';
```

运行结果

```
hive> -- 创建数据表
hive> create table student(Sno int,Sname string,Sex string,Sage int,Sdept string)
    > row format delimited fields terminated by ',' stored as textfile;
OK
Time taken: 0.137 seconds
hive> create table course(Cno int,Cname string)
    > row format delimited fields terminated by ',' stored as textfile;
OK
Time taken: 0.112 seconds
hive> create table sc(Sno int,Cno int,Grade int)
    > row format delimited fields terminated by ',' stored as textfile;
OK
Time taken: 0.061 seconds
hive>
hive> -- 向数据表中装载文件
hive> load data inpath '/user/myname/stu/students.txt' overwrite into table student;
Loading data to table student.student
OK
Time taken: 0.376 seconds
hive>
hive> load data inpath '/user/myname/stu/course.txt' overwrite into table course;
Loading data to table student.course
OK
Time taken: 0.252 seconds
hive> load data inpath '/user/myname/stu/sc.txt' overwrite into table sc;
Loading data to table student.sc
OK
Time taken: 0.276 seconds
hive>
hive> -- 查询数据
hive> select Sno,Sname from student;
OK
2018213201      李小勇
2018213202      刘良
2018213203      王芝芝
2018213204      张大立
2018213205      刘云山
2018213206      孙庆刚
2018213207      易思玲
2018213208      李娜
2018213209      梦媛媛
2018213210      孔江涛
2018213211      包小波
2018213212      孙晓花
2018213213      冯伟伟
2018213214      王小容
2018213216      王君丹
2018213215      钱国峰
```

```
2018213217      王风娟
2018213218      王一一
2018213219      邢小雨
2018213220      赵钱钱
2018213221      周小二
2018213222      郑明山
Time taken: 0.134 seconds, Fetched: 22 row(s)
hive> select Sno,Sname from student where Sex='男';
OK
2018213201      李小勇
2018213204      张大立
2018213205      刘云山
2018213206      孙庆刚
2018213210      孔江涛
2018213211      包小波
2018213213      冯伟伟
2018213216      王君丹
2018213215      钱国峰
2018213220      赵钱钱
2018213221      周小二
2018213222      郑明山
Time taken: 0.139 seconds, Fetched: 12 row(s)
hive> select distinct Sname from student inner join sc on
student.Sno=sc.Sno;
Query ID = root_20231109140019_1e25a730-12e3-439b-9c74-0fdadb7b6244
Total jobs = 1
2023-11-09 14:00:30     Dump the side-table for tag: 0 with group count: 22
into file: file:/tmp/root/6c7f3d1e-927d-4f70-88e4-09aeca3d0443/hive_2023-
11-09_14-00-19_828_5670865665580644800-1/-local-10005/HashTable-Stage-
2/MapJoin-mapfile00--.hashtable
2023-11-09 14:00:30     Uploaded 1 File to: file:/tmp/root/6c7f3d1e-927d-
4f70-88e4-09aeca3d0443/hive_2023-11-09_14-00-19_828_5670865665580644800-1/-
local-10005/HashTable-Stage-2/MapJoin-mapfile00--.hashtable (983 bytes)
2023-11-09 14:00:30     End of local task; Time Taken: 1.637 sec.
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1699491849504_0001, Tracking URL =
http://master:8088/proxy/application_1699491849504_0001/
Kill Command = /usr/local/hadoop-3.1.4/bin/mapred job  -kill
job_1699491849504_0001
Hadoop job information for Stage-2: number of mappers: 1; number of
reducers: 1
2023-11-09 14:00:43,702 Stage-2 map = 0%,  reduce = 0%
2023-11-09 14:00:54,068 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU
2.85 sec
2023-11-09 14:01:00,215 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU
4.83 sec
```

```
MapReduce Total cumulative CPU time: 4 seconds 830 msec
Ended Job = job_1699491849504_0001
MapReduce Jobs Launched:
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 4.83 sec   HDFS Read: 18207 HDFS Write: 757 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 830 msec
OK
冯伟伟
刘云山
刘良
包小波
周小二
孔江涛
孙庆刚
孙晓花
张大立
易思玲
李娜
李小勇
梦媛媛
王一一
王君丹
王小容
王芝芝
王风娟
赵钱钱
邢小雨
郑明山
钱国峰
Time taken: 42.522 seconds, Fetched: 22 row(s)
hive>
hive> -- 查询数据，查询学生的课程成绩情况
hive> select student.Sname,Course.Cname,Sc.Grade from student join sc
    > on student.Sno=sc.Sno join course on sc.cno=course.cno;
No Stats for student@student, Columns: sno, sname
No Stats for student@sc, Columns: sno, cno, grade
No Stats for student@course, Columns: cno, cname
Query ID = root_20231109140520_f5da6c0d-77b7-4fbe-ad10-06cdb071c284
Total jobs = 1
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1699491849504_0002, Tracking URL = http://master:8088/proxy/application_1699491849504_0002/
Kill Command = /usr/local/hadoop-3.1.4/bin/mapred job  -kill job_1699491849504_0002
Hadoop job information for Stage-5: number of mappers: 1; number of reducers: 0
2023-11-09 14:05:39,581 Stage-5 map = 0%,  reduce = 0%
2023-11-09 14:05:46,821 Stage-5 map = 100%,  reduce = 0%, Cumulative CPU 2.2 sec
MapReduce Total cumulative CPU time: 2 seconds 200 msec
Ended Job = job_1699491849504_0002
MapReduce Jobs Launched:
```

```
Stage-Stage-5: Map: 1   Cumulative CPU: 2.2 sec   HDFS Read: 12571 HDFS Write: 6514 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 200 msec
OK
李小勇  Python语言程序设计      81
李小勇  数据库系统原理  85
李小勇  信息系统分析与设计      88
李小勇  大数据技术及应用        70
刘良    数据库系统原理  90
刘良    信息系统分析与设计      80
刘良    大数据技术及应用        71
刘良    Hadoop开发与应用        60
王芝芝  Python语言程序设计      82
王芝芝  信息系统分析与设计      90
王芝芝  Hadoop开发与应用        100
张大立  Python语言程序设计      80
张大立  数据库系统原理  92
张大立  大数据技术及应用        91
张大立  Hadoop开发与应用        70
刘云山  Python语言程序设计      70
刘云山  数据库系统原理  92
刘云山  信息系统分析与设计      99
刘云山  Web应用程序设计 87
孙庆刚  Python语言程序设计      72
孙庆刚  数据库系统原理  62
孙庆刚  信息系统分析与设计      100
孙庆刚  大数据技术及应用        59
孙庆刚  Hadoop开发与应用        60
孙庆刚  Web应用程序设计 98
易思玲  信息系统分析与设计      68
易思玲  大数据技术及应用        91
易思玲  Hadoop开发与应用        94
易思玲  Web应用程序设计 78
李娜    Python语言程序设计      98
李娜    信息系统分析与设计      89
李娜    Web应用程序设计 91
梦媛媛  数据库系统原理  81
梦媛媛  大数据技术及应用        89
梦媛媛  Web应用程序设计 100
孔江涛  数据库系统原理  98
孔江涛  Hadoop开发与应用        90
孔江涛  Web应用程序设计 80
包小波  Python语言程序设计      81
包小波  数据库系统原理  91
包小波  信息系统分析与设计      81
包小波  大数据技术及应用        86
孙晓花  Python语言程序设计      81
孙晓花  信息系统分析与设计      78
孙晓花  大数据技术及应用        85
孙晓花  Web应用程序设计 98
冯伟伟  Python语言程序设计      98
冯伟伟  数据库系统原理  58
冯伟伟  大数据技术及应用        88
冯伟伟  Hadoop开发与应用        93
王小容  Python语言程序设计      91
```

```
王小容  数据库系统原理  100
王小容  大数据技术及应用  98
钱国峰  Python语言程序设计  91
钱国峰  信息系统分析与设计  59
钱国峰  大数据技术及应用  100
钱国峰  Web应用程序设计  95
王君丹  Python语言程序设计  92
王君丹  数据库系统原理  99
王君丹  大数据技术及应用  82
王风娟  大数据技术及应用  82
王风娟  Hadoop开发与应用  100
王风娟  Web应用程序设计  58
王一一  Python语言程序设计  95
王一一  数据库系统原理  100
王一一  信息系统分析与设计  67
王一一  大数据技术及应用  78
邢小雨  Python语言程序设计  77
邢小雨  数据库系统原理  90
邢小雨  信息系统分析与设计  91
邢小雨  大数据技术及应用  67
邢小雨  Hadoop开发与应用  87
赵钱钱  Python语言程序设计  66
赵钱钱  数据库系统原理  99
赵钱钱  Hadoop开发与应用  93
周小二  数据库系统原理  93
周小二  Hadoop开发与应用  91
周小二  Web应用程序设计  99
郑明山  信息系统分析与设计  69
郑明山  大数据技术及应用  93
郑明山  Hadoop开发与应用  82
郑明山  Web应用程序设计  100
Time taken: 28.546 seconds, Fetched: 82 row(s)
hive> -- 查询数据，查询选修2号课程且成绩为90分以上的学生
hive> select student.Sname,sc.Grade from student join sc
    > on student.Sno=sc.Sno and sc.Grade>90;
Query ID = root_20231109140552_8e9aa7db-e4c9-477e-8357-75851f1d8546
Total jobs = 1
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1699491849504_0003, Tracking URL =
http://master:8088/proxy/application_1699491849504_0003/
Kill Command = /usr/local/hadoop-3.1.4/bin/mapred job  -kill
job_1699491849504_0003
Hadoop job information for Stage-3: number of mappers: 1; number of
reducers: 0
2023-11-09 14:06:09,646 Stage-3 map = 0%,  reduce = 0%
2023-11-09 14:06:16,840 Stage-3 map = 100%,  reduce = 0%, Cumulative CPU
2.13 sec
MapReduce Total cumulative CPU time: 2 seconds 130 msec
Ended Job = job_1699491849504_0003
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 2.13 sec   HDFS Read: 10684 HDFS
Write: 1307 SUCCESS
```

```
Total MapReduce CPU Time Spent: 2 seconds 130 msec
OK
王芝芝  100
张大立  92
张大立  91
刘云山  92
刘云山  99
孙庆刚  100
孙庆刚  98
易思玲  91
易思玲  94
李娜    98
李娜    91
梦媛媛  100
孔江涛  98
包小波  91
孙晓花  98
冯伟伟  98
冯伟伟  93
王小容  91
王小容  100
王小容  98
钱国峰  91
钱国峰  100
钱国峰  95
王君丹  92
王君丹  99
王风娟  100
王一一  95
王一一  100
邢小雨  91
赵钱钱  99
赵钱钱  93
周小二  93
周小二  91
周小二  99
郑明山  93
郑明山  100
Time taken: 26.779 seconds, Fetched: 36 row(s)
hive>
hive> -- 查询数据，查询各个课程及相应的选课人数
hive> select Cno,count(1) from sc group by Cno;
Query ID = root_20231109140754_ec86bbd4-f559-4fe9-803f-d76373782bab
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1699491849504_0004, Tracking URL =
http://master:8088/proxy/application_1699491849504_0004/
```

```
Kill Command = /usr/local/hadoop-3.1.4/bin/mapred job  -kill job_1699491849504_0004
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2023-11-09 14:08:01,568 Stage-1 map = 0%,  reduce = 0%
2023-11-09 14:08:09,757 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.76 sec
2023-11-09 14:08:15,863 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.64 sec
MapReduce Total cumulative CPU time: 3 seconds 640 msec
Ended Job = job_1699491849504_0004
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.64 sec   HDFS Read: 14115 HDFS Write: 189 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 640 msec
OK
1       15
2       15
3       13
4       16
5       12
6       11
Time taken: 22.935 seconds, Fetched: 6 row(s)
hive>
hive> -- 查询数据，查询选修了3门以上的课程的学生学号
hive> select Sno from sc group by Sno having count(Cno)>3;
Query ID = root_20231109141159_fd63bf1b-2aff-4821-816a-0bb649364ac0
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1699491849504_0005, Tracking URL = http://master:8088/proxy/application_1699491849504_0005/
Kill Command = /usr/local/hadoop-3.1.4/bin/mapred job  -kill job_1699491849504_0005
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2023-11-09 14:12:07,476 Stage-1 map = 0%,  reduce = 0%
2023-11-09 14:12:15,651 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.75 sec
2023-11-09 14:12:21,772 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.39 sec
MapReduce Total cumulative CPU time: 4 seconds 390 msec
Ended Job = job_1699491849504_0005
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.39 sec   HDFS Read: 14915 HDFS Write: 386 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 390 msec
OK
2018213201
```

```
2018213202
2018213204
2018213205
2018213206
2018213207
2018213211
2018213212
2018213213
2018213215
2018213218
2018213219
2018213222
Time taken: 23.127 seconds, Fetched: 13 row(s)
hive>
hive> -- 查询数据，查询学生信息并根据学号进行升序排序
hive> select Sno from student order by Sno;
Query ID = root_20231109141356_e7a8733b-9cca-4366-9f4f-c3d3b1a64e9e
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1699491849504_0006, Tracking URL =
http://master:8088/proxy/application_1699491849504_0006/
Kill Command = /usr/local/hadoop-3.1.4/bin/mapred job  -kill
job_1699491849504_0006
Hadoop job information for Stage-1: number of mappers: 1; number of
reducers: 1
2023-11-09 14:14:04,015 Stage-1 map = 0%,  reduce = 0%
2023-11-09 14:14:11,289 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU
1.73 sec
2023-11-09 14:14:17,572 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU
3.59 sec
MapReduce Total cumulative CPU time: 3 seconds 590 msec
Ended Job = job_1699491849504_0006
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.59 sec   HDFS Read:
10624 HDFS Write: 593 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 590 msec
OK
2018213201
2018213202
2018213203
2018213204
2018213205
2018213206
2018213207
2018213208
2018213209
2018213210
2018213211
```

```
2018213212
2018213213
2018213214
2018213215
2018213216
2018213217
2018213218
2018213219
2018213220
2018213221
2018213222
Time taken: 23.267 seconds, Fetched: 22 row(s)
hive>
hive> -- 查询数据，按性别分区，在分区内按年龄升序排序
hive> set mapred.reduce.tasks=2;
hive> insert overwrite directory '/user/myname/stu_out'
    > select * from student
    > distribute by Sex sort by Sage;
Query ID = root_20231109143653_331e4547-4f79-4ca1-b68e-fb348d8e7b56
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Defaulting to jobconf value of: 2
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1699491849504_0007, Tracking URL =
http://master:8088/proxy/application_1699491849504_0007/
Kill Command = /usr/local/hadoop-3.1.4/bin/mapred job  -kill
job_1699491849504_0007
Hadoop job information for Stage-1: number of mappers: 1; number of
reducers: 2
2023-11-09 14:37:00,753 Stage-1 map = 0%,  reduce = 0%
2023-11-09 14:37:08,011 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU
1.82 sec
2023-11-09 14:37:14,130 Stage-1 map = 100%,  reduce = 50%, Cumulative CPU
3.84 sec
2023-11-09 14:37:18,208 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU
5.6 sec
MapReduce Total cumulative CPU time: 5 seconds 600 msec
Ended Job = job_1699491849504_0007
Moving data to directory /user/myname/stu_out
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 2   Cumulative CPU: 5.6 sec   HDFS Read:
16825 HDFS Write: 676 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 600 msec
OK
Time taken: 25.769 seconds
hive>
hive> -- 插入数据
hive> -- 插入数据,在student表中插入一条新记录
hive> insert into table student values(2018213223,'王小哲','男',18,'IS');
Query ID = root_20231109144034_0053ca86-2fa4-4629-831e-8c7196b41375
```

```
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1699491849504_0008, Tracking URL =
http://master:8088/proxy/application_1699491849504_0008/
Kill Command = /usr/local/hadoop-3.1.4/bin/mapred job  -kill
job_1699491849504_0008
Hadoop job information for Stage-1: number of mappers: 1; number of
reducers: 1
2023-11-09 14:40:42,419 Stage-1 map = 0%,  reduce = 0%
2023-11-09 14:40:50,592 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU
2.34 sec
2023-11-09 14:40:56,725 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU
3.91 sec
MapReduce Total cumulative CPU time: 3 seconds 910 msec
Ended Job = job_1699491849504_0008
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory
hdfs://master:8020/user/hive_remote/warehouse/student.db/student/.hive-
staging_hive_2023-11-09_14-40-34_595_847354642008985408-1/-ext-10000
Loading data to table student.student
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.91 sec   HDFS Read:
18710 HDFS Write: 384 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 910 msec
OK
Time taken: 24.831 seconds
hive> -- 删除数据
hive> delete from student where Sno='2018213223';
FAILED: SemanticException [Error 10294]: Attempt to do update or delete
using transaction manager that does not support these operations.
hive>
```

注：默认hive不支持记录的删除，若要使用update和delete删除操作，需要配置hive-site.xml，这里省略。

任务6.5掉线率top10基站统计

自我国三大运营商获得无线牌照以来，无线用户发展迅猛，优化维护工作的好坏直接影响运营商的服务质量和满意度。而影响运营商服务质量和满意度的一个重要网络指标之一——掉话率。因此，统计分析各基站掉话率，有助于运营商做出科学决策，提升网络质量，开展优化维护工作，降低基站掉话率，才能有效的支撑业务发展，提升用户满足度。本小节的任务如下。统计出每个基站的掉话率，并按降序排序。找出掉话率比较高的前20个基站。

分析基本思路

根据jizhan_information.scv文件中的信息，计算每个基站的掉话率，找出最高掉话率的基站，可以帮助运营商更好的分析高掉话率基站的具体情况，有助于安排维护人员有针对性的进行故障检测。已知基站掉话率的计算公式，如下图所示。

$$掉话率 = \frac{掉话总时长}{通话持续总时长}$$

统计基站的掉话率的实现步骤如下:

1. 上传jizhan_information.csv文件至服务器。
2. 创建myhive数据库并在myhive数据库中创建jizhan结构表。
3. 装载jizhan_information.scv文件中的数据至jizhan表。
4. 创建jizhan_result结果表，用以存储统计掉话率后的信息。
5. 根据掉话率公式统计各基站的掉话率，并按降序排序，找出掉话率Top20个基站。

任务实现

数据文件准备

如何本章数据已完成执行，这里可以忽略

```
[root@master ~]# cd /root/hadoop
[root@master hadoop]# hdfs dfs -put ./hadoop_data/jizhan_information.csv /user/myname/stu
[root@master hadoop]#
```

数据操作

在master或slave上执行hive进入hive命令行操作:

```
show databases;
use student;
create table jizhan(
record_time string,
imei int,
cell string,
ph_num int,
call_num int,
drop_num int,
duration int,
drop_rate double,
net_type string,
er1 int) row format delimited fields terminated by ',';
-- 装载数据文件
load data inpath '/user/myname/stu/jizhan_information.csv' into table jizhan;
-- 查询已装载的数据表内容
select * from jizhan limit 10;
-- 创建结果表
create table jizhan_result(
imei string,
drop_num int,
duration int,
```

```
drop_rate double);
-- 统计基站掉话率
from jizhan
insert into jizhan_result
select imei,sum(drop_num) as sdrop,sum(duration) as sdura,
sum(drop_num)/sum(duration) as drop_rate
group by imei
order by drop_rate desc;
-- 查询统计结果
select * from jizhan_result limit 10;
```

运行结果如：

```
hive> show databases;
OK
default
hive_remote
student
Time taken: 0.071 seconds, Fetched: 3 row(s)
hive> use student;
OK
Time taken: 0.036 seconds
hive>
hive> create table jizhan(
    > record_time string,
    > imei int,
    > cell string,
    > ph_num int,
    > call_num int,
    > drop_num int,
    > duration int,
    > drop_rate double,
    > net_type string,
    > er1 int) row format delimited fields terminated by ',';
OK
Time taken: 0.146 seconds
hive>
hive> -- 装载数据文件
hive> load data inpath '/user/myname/stu/jizhan_information.csv' into table
jizhan;
Loading data to table student.jizhan
OK
Time taken: 0.217 seconds
hive> -- 查询已装载的数据表内容
hive> select * from jizhan limit 10;
OK
record_time     NULL    cell    NULL    NULL    NULL    NULL    NULL
net_type        NULL
2011-07-13 00:00:00+08  356966  29448-37062     0       0       0       0
    0.0     G       0
2011-07-13 00:00:00+08  352024  29448-51331     0       0       0       0
    0.0     G       0
2011-07-13 00:00:00+08  353736  29448-51331     0       0       0       0
    0.0     G       0
```

```
2011-07-13 00:00:00+08  353736  29448-51333     0       0       0       0
    0.0     G       0
2011-07-13 00:00:00+08  351545  29448-51333     0       0       0       0
    0.0     G       0
2011-07-13 00:00:00+08  353736  29448-51343     1       0       0       8
    0.0     G       0
2011-07-13 00:00:00+08  359681  29448-51462     0       0       0       0
    0.0     G       0
2011-07-13 00:00:00+08  354707  29448-51462     0       0       0       0
    0.0     G       0
2011-07-13 00:00:00+08  356137  29448-51470     0       0       0       0
    0.0     G       0
Time taken: 0.164 seconds, Fetched: 10 row(s)
hive>
hive> -- 统计基站掉话率
hive> from jizhan
    > insert into jizhan_result
    > select imei,sum(drop_num) as sdrop,sum(duration) as sdura,
    > sum(drop_num)/sum(duration) as drop_rate
    > group by imei
    > order by drop_rate desc;
Query ID = root_20231109153639_a1df8724-256c-4dae-8834-fd1151de7eb3
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Defaulting to jobconf value of: 2
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1699491849504_0009, Tracking URL =
http://master:8088/proxy/application_1699491849504_0009/
Kill Command = /usr/local/hadoop-3.1.4/bin/mapred job  -kill
job_1699491849504_0009
Hadoop job information for Stage-1: number of mappers: 1; number of
reducers: 2
2023-11-09 15:36:48,100 Stage-1 map = 0%,  reduce = 0%
2023-11-09 15:36:56,600 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU
3.82 sec
2023-11-09 15:37:03,013 Stage-1 map = 100%,  reduce = 50%, Cumulative CPU
6.58 sec
2023-11-09 15:37:08,368 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU
9.47 sec
MapReduce Total cumulative CPU time: 9 seconds 470 msec
Ended Job = job_1699491849504_0009
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
```

```
Starting Job = job_1699491849504_0010, Tracking URL =
http://master:8088/proxy/application_1699491849504_0010/
Kill Command = /usr/local/hadoop-3.1.4/bin/mapred job  -kill
job_1699491849504_0010
Hadoop job information for Stage-2: number of mappers: 1; number of
reducers: 1
2023-11-09 15:37:21,675 Stage-2 map = 0%,  reduce = 0%
2023-11-09 15:37:29,831 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU
2.01 sec
2023-11-09 15:37:35,982 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU
5.37 sec
MapReduce Total cumulative CPU time: 5 seconds 370 msec
Ended Job = job_1699491849504_0010
Loading data to table student.jizhan_result
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 2   Cumulative CPU: 9.47 sec   HDFS Read:
57423621 HDFS Write: 380103 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 5.37 sec   HDFS Read:
394032 HDFS Write: 264283 SUCCESS
Total MapReduce CPU Time Spent: 14 seconds 840 msec
OK
Time taken: 58.821 seconds
hive>
hive> -- 查询统计结果
hive> select * from jizhan_result limit 10;
OK
639876  1       734     0.0013623978201634877
356436  1       1028    9.727626459143969E-4
351760  1       1232    8.116883116883117E-4
368883  1       1448    6.906077348066298E-4
358849  1       1469    6.807351940095302E-4
358231  1       1613    6.199628022318661E-4
863738  2       3343    5.982650314089142E-4
865011  1       1864    5.36480686695279E-4
862242  1       1913    5.227391531625719E-4
350301  2       3998    5.002501250625312E-4
Time taken: 0.107 seconds, Fetched: 10 row(s)
hive>
```

小结

本章详细介绍了Hive的基本知识。Hive是一个构建在Hadoop之上的数据仓库工具，主要用于对存储在Hadoop文件中的数据集进行数据整理、特殊查询和分析处理。本章从Hive与传统数据库的区别出发，通过介绍Hive的基本概念，让读者需要了解Hive以及Hive架构、数据模型及其工作原理；通过介绍Hive的3种访问方式及搭建过程，让读者熟悉Hive的安装步骤和管理。接着介绍Hive的数据操作，让读者掌握HiveQL的相关操作。作为初学者，学习Hive需要实际动手操作Hive，这也是掌握Hive的关键，最后以掉线率Top20基站统计为例，详细介绍了如何使用Hive解决具体的实际问题。

版本历史

- Ver1.0-20231109

- 初始版本

上一章节 下一章节