

Hive 课程讲义

Hive 课程讲义.....	1
第 1 次课 Hive 入门.....	4
1.1. 教学目标.....	4
1.1.1. 前言.....	4
1.1.2. 掌握 Hive 的是什么.....	4
1.1.3. 掌握 Hive 的体系结构.....	4
1.1.4. 理解 Hive 的元数据 Metastore 的存储及其作用.....	4
1.1.5. 掌握 Mysql 和 Hive 的安装.....	4
1.2. 课程内容.....	4
1.2.1. 前言.....	4
1.2.2. Hive 是什么.....	5
1.2.3. Hive 的系统结构和数据存储.....	5
1.2.3.1. Hive 的体系结构.....	5
1.2.3.2. Hive 的数据存储.....	6
1.2.4. Hive 的元数据 Metastore.....	6
1.2.5. Hive 的安装与运行.....	7
1.2.5.1. Mysql 安装.....	7
离线安装:	7
在线安装.....	10
1.2.5.2. Hive 的安装.....	11
1.2.5.3. Hive 的运行及访问方式.....	13
1.3. 考题.....	16
1.4. 实验.....	16
1.4.1. 搭建 Hive 开发环境.....	16
1.4.1.1. 环境.....	16
1.4.1.2. 步骤.....	17
1.4.1.3. 验证.....	18
第 2 次课 Hive 简单操作（一）.....	20
2.1. 教学目标.....	20
2.1.1. 了解 Hive 日志信息.....	20
2.1.2. 掌握数据库相关概念和操作.....	20
2.2. 课程内容.....	20
2.2.1. Hive 日志信息.....	20
2.2.1.1. 去掉 Hive 启动时候的日志警告信息.....	20
2.2.1.2. Hive 的日志.....	20
2.2.2. Hive 中的数据库、表、数据与 HDFS 的对应关系.....	22
2.2.2.1. Hive 数据库.....	22
2.2.2.1.1. 查看并使用数据库.....	22
2.2.2.1.2. 创建数据库.....	23

2.2.2.1.3. 删除数据库.....	24
2.2.2.2. Hive 数据类型.....	25
2.2.2.3. Hive 表.....	26
2.2.2.3.1. 创建表.....	26
2.2.2.3.2. 查看所有的表.....	26
2.2.2.3.3. 查看表结构.....	26
2.2.2.3.4. 表在 hdfs 中的位置.....	27
2.2.2.3.5. 表在元数据 metastore 中的体现.....	27
2.2.2.3.6. 加载数据到表中的两种方式.....	28
2.2.2.3.7. 数据加载的两种模式.....	29
2.2.2.4. 修改表，表的常用 DDL.....	30
2.2.2.4.1. 重命名表明.....	30
2.2.2.4.2. 给表增加一个字段：.....	30
2.2.2.4.3. 修改某一个字段：.....	31
2.2.2.4.4. Replace 替换：.....	32
2.2.2.5. 列分隔符.....	32
2.2.2.6. 复杂数据类型.....	35
2.2.2.6.1. array.....	35
2.2.2.6.2. map.....	36
2.2.2.6.3. struct.....	38
2.2.2.6.4. 综合案例.....	39
2.3. 考题.....	40
2.4. 实验.....	40
第 3 次课 Hive 操作（二）.....	42
3.1. 教学目标.....	42
3.1.1. 掌握 Hive 表的类型.....	42
3.1.2. 掌握数据加载语句.....	42
3.1.3. 掌握数据的导出.....	42
3.2. 课程内容.....	42
3.2.1. Hive 表的类型.....	42
3.2.1.1. 受控表.....	42
3.2.1.2. 外部表.....	42
3.2.1.3. 分区表.....	45
3.2.1.3.1. 分区表的 CRUD.....	45
3.2.1.3.2. 关联 HDFS 中的数据到分区中.....	49
3.2.1.4. 桶表.....	50
3.2.2. 视图.....	52
3.2.3. 索引.....	54
3.2.4. 数据加载语句.....	54
3.2.4.1. 从文件中装载.....	54
3.2.4.2. 从其他表中装载.....	57
3.2.4.3. 通过动态分区装载数据.....	58
3.2.4.4. 建立表的同时装载数据.....	58
3.2.5. 数据导出.....	59

3.2.6. 本地模式.....	59
3.3. 考题.....	60
3.4. 实验.....	60
第 4 次课 Hive 高阶查询.....	60
4.1. 教学目标.....	60
4.1.1. Hive 的查询条件.....	60
4.1.2. Hive 中的 MapReduce 任务.....	60
4.1.3. 控制 Hive 中 MR 数量.....	60
4.1.4. Hive 中排序.....	60
4.2. 课程内容.....	60
4.2.1. 条件显示.....	60
4.2.2. Hive 中的 MapReduce 任务.....	62
4.2.2.1. 什么情况下 Hive 可以避免 MR.....	62
4.2.2.2. MR 的执行过程.....	62
4.2.3. 控制 Hive 中 MR 数量.....	63
4.2.3.1. 控制 Map Task 的数量.....	63
4.2.3.2. 控制 Reduce Task 的数量.....	64
4.2.4. reduce 数量只有一个.....	64
4.2.5. Hive 中排序.....	65
4.3. 考题.....	66
4.4. 实验.....	66
第 5 次课 Hive 高阶应用——函数.....	67
5.1. 教学目标.....	67
5.1.1. Hive 中数据的存储类型.....	67
5.1.2. Hive 中的内嵌函数.....	67
5.1.3. Hive 自定义函数.....	67
5.1.4. Hive JDBC.....	67
5.2. 课程内容.....	67
5.2.1. Hive 中文件的存储类型.....	67
5.2.1.1. TextFile.....	67
5.2.1.2. SequenceFile.....	68
5.2.1.3. RCFile.....	69
5.2.1.4. 总结.....	69
5.2.2. Hive 中的内嵌函数.....	70
5.2.3. Hive 自定义函数.....	75
5.2.4. Hive JDBC.....	77
5.3. 考题.....	81
5.4. 实验.....	81
附件说明：	82
1、将普通用户赋予 sudo 的权限.....	82
2、MySQL 编码设置.....	83
2.1、离线安装 mysql 的修改方式.....	83
2.2、在线安装 mysql 的修改方式.....	84
3、Hive 参数说明.....	86

4、字段定义时的 Comment.....	86
5、关于 Sequence 不支持 GzipCodec.....	86

第 1 次课 Hive 入门

1. 1. 教学目标

1. 1. 1. 前言

1. 1. 2. 掌握 Hive 的是什么

1. 1. 3. 掌握 Hive 的体系结构

1. 1. 4. 理解 Hive 的元数据 Metastore 的存储及其作用

1. 1. 5. 掌握 Mysql 和 Hive 的安装

1. 2. 课程内容

1. 2. 1. 前言

我们知道，我们对 HDFS 的访问可以有三种方式，第一种通过 CLI 命令行 shell 方式来访问；第二种，可以通过 java 编程来访问；第三种，可以通过 web 来访问。这三种的访问方式呢，每一种都或多或少有局限，其实是可以满足我们日常的统计分析需要，但是我们老是觉得，还是有那么一点点遗憾，我们来看一下。

一般的，要想接触 hadoop 大数据处理方案，先苦学一堆 java 相关知识，耗费了大量的时间；开发 mapreduce 同样，虽然其运行效率相对较高，但是开发 mapreduce 需要编码，测试、部署等等一些列阶段，也耗费了不少时间，这对我们的开发人员实际上是一种折磨，对开发资源也是一种浪费，明明可以 1 个小时干完的事，结果做了 1 天。

实际上呢，我们使用最频繁的统计分析工具或是软件，就是 SQL，如果能有类似的东西来满足于我们在海量数据中的分析统计的话，那会是使得我们学习、开发成本大为降低，开发效率提高。我们学习 php、asp 等等其他的程序员也可以来玩大数据，因为业界的 SQL

是标准，统一，每个软件几乎都要用。

实际上呢，专门就有这样的产品，那就是 Apache-Hive，Hadoop 生态链中非常重要的顶级项目！

1.2.2. Hive 是什么



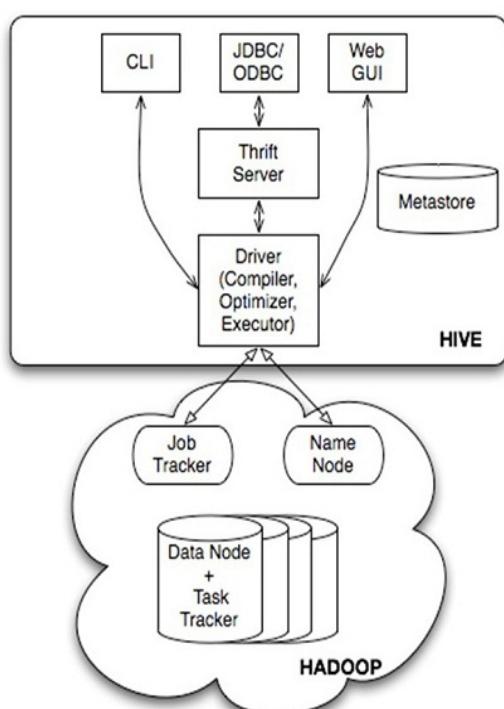
Hive 是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL），这是一种可以存储、查询和分析存储在 Hadoop 中的海量数据的机制。Hive 定义了简单的类 SQL 查询语言，称为 HQL，它允许熟悉 SQL 的用户查询数据。同时，这个语言也允许熟悉 MapReduce 开发者的开发自定义的 mapper 和 reducer 来处理内建的 mapper 和 reducer 无法完成的复杂的分析工作。

Hive 是 SQL 解析引擎，它将 SQL 语句转译成 M/R Job 然后在 Hadoop 执行。

Hive 的表其实就是 HDFS 的目录，按表名把文件夹分开。如果是分区表，则分区值是子文件夹，可以直接在 M/R Job 里使用这些数据。

Hive 相当于 hadoop 的客户端工具，部署时不一定放在集群管理节点中，可以放在某个节点上。

1.2.3. Hive 的系统结构和数据存储



1.2.3.1. Hive 的体系结构

- 用户接口，包括 CLI、JDBC/ODBC、WebGUI

CLI，即 Shell 命令行

JDBC/ODBC 是 Hive 的 Java，与使用传统数据库 JDBC 的方式类似

WebGUI 是通过浏览器访问

Hive

- 元数据存储，通常是存储在关系数据库如 mysql, derby 中

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

- 解释器、编译器、优化器、执行器

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

- Hadoop：用 HDFS 进行存储，利用 MapReduce 进行计算

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

1.2.3.2. Hive 的数据存储

Hive 的数据存储基于 Hadoop HDFS

Hive 没有专门的数据存储格式

存储结构主要包括：数据库、文件、表、视图、索引

Hive 默认可以直接加载文本文件 (TextFile)，还支持 SequenceFile、RCFile

创建表时，指定 Hive 数据的列分隔符与行分隔符，Hive 即可解析数据

1.2.4. Hive 的元数据 Metastore

metastore 是 hive 元数据的集中存放地。

metastore 默认使用内嵌的 derby 数据库作为存储引擎，目前只支持 derby 和 mysql，作为存储引擎。

Derby 引擎的缺点：一次只能打开一个会话

使用 MySQL 作为外置存储引擎，多用户同时访问

1.2.5. Hive 的安装与运行

在这里呢，我们采用 mysql 作为存储引擎。所以安装 Hive 的第一步，安装 Mysql，

这里呢，提供两种方式安装 Mysql，离线和在线。

1.2.5.1. Mysql 安装

特别说明：在下面的安装过程中我是用普通用户来安装的，它是需要 root 权限的，所以必须要用 sudo 来执行，大家用 root 用户的话凡是我加 root 的地方，把 sudo 去掉就可以了，可以跳过特别说明这一步。

如果想在普通用户下安装，默认是无法执行 sudo 的，如下图错误，需要修改一下 /etc/sudoers 文件，讲我们的相应的普通用户加入到里面（操作过程在 root 用户下执行）。

```
[crxy@master ~]$ sudo rpm -qa | grep mysql
[sudo] password for crxy:
crxy is not in the sudoers file. This incident will be reported.
```

如何修改？[点我](#)（按住 ctrl 键，然后鼠标左键）！

离线安装：

如图所示为 Mysql 离线安装包

```
[user@crxy soft]$ ll
总用量 23624
-rw-rw-r-- 1 user user 7412135 12月 24 07:26 MySQL-client-5.1.73-1.glibc23.x86_64.rpm
-rw-rw-r-- 1 user user 16775717 12月 24 07:26 MySQL-server-5.1.73-1.glibc23.x86_64.rpm
[user@crxy soft]$ sudo rpm -qa | grep mysql
[sudo] password for user:
[user@crxy soft]$
```

第一步：删除 mysql 的依赖

```
rpm -qa | grep mysql 或者 (MySQL)
sudo rpm -e --nodeps `rpm -qa | grep MySQL`
```

```
[crxy@hive soft]$ rpm -qa | grep MySQL
MySQL-server-5.1.73-1.glibc23.x86_64
MySQL-client-5.1.73-1.glibc23.x86_64
perl-DBD-MySQL-4.013-3.el6.x86_64
[crxy@hive soft]$ sudo rpm -e --nodeps `rpm -qa | grep MySQL`
[sudo] password for crxy:
[crxy@hive soft]$ rpm -qa | grep MySQL
```

第二步：安装 mysql

使用命令 `sudo rpm -ivh MySQL-server-5.1.73-1.glibc23.x86_64.rpm`

```
[user@crxy soft]$ sudo rpm -ivh MySQL-server-5.1.73-1.glibc23.x86_64.rpm
Preparing...                                              [100%]
 1:MySQL-server                                         [100%]

PLEASE REMEMBER TO SET A PASSWORD FOR THE MySQL root USER !
To do so, start the server, then issue the following commands:

/usr/bin/mysqladmin -u root password 'new-password'
/usr/bin/mysqladmin -u root -h crxy password 'new-password'

Alternatively you can run:
/usr/bin/mysql_secure_installation

which will also give you the option of removing the test
databases and anonymous user created by default. This is
strongly recommended for production servers.

See the manual for more instructions.

Please report any problems with the /usr/bin/mysqlbug script!

Starting MySQL..... SUCCESS!
```

使用命令 `sudo rpm -ivh MySQL-client-5.1.73-1.glibc23.x86_64.rpm`

```
[user@crxy soft]$ sudo rpm -ivh MySQL-client-5.1.73-1.glibc23.x86_64.rpm
Preparing...                                              [100%]
 1:MySQL-client                                         [100%]
```

初次安装成功之后，会出现上述的结果，这里执行 `mysql_secure_installation` 脚本进行初始化设置。

说明：如果不是以前安装过 mysql，在安装的是不会有上述的安装 mysql server 显示的图片效果的，需要做一个步骤讲 mysql 服务启动

```
sudo service mysql start
```

```
[user@crxy soft]$ /usr/bin/mysql_secure_installation

NOTE: RUNNING ALL PARTS OF THIS SCRIPT IS RECOMMENDED FOR ALL MySQL
      SERVERS IN PRODUCTION USE! PLEASE READ EACH STEP CAREFULLY!

In order to log into MySQL to secure it, we'll need the current
password for the root user. If you've just installed MySQL, and
you haven't set the root password yet, the password will be blank,
so you should just press enter here.

Enter current password for root (enter for none):
OK, successfully used password, moving on...

Setting the root password ensures that nobody can log into the MySQL
root user without the proper authorisation.

Set root password? [Y/n] Y
New password:
Re-enter new password:
Password updated successfully!
Reloading privilege tables..
... Success!

By default, a MySQL installation has an anonymous user, allowing anyone
to log into MySQL without having to have a user account created for
them. This is intended only for testing, and to make the installation
go a bit smoother. You should remove them before moving into a
production environment.

Remove anonymous users? [Y/n] Y
... Success!

Normally, root should only be allowed to connect from 'localhost'. This
ensures that someone cannot guess at the root password from the network.

Disallow root login remotely? [Y/n] Y
... Success!

By default, MySQL comes with a database named 'test' that anyone can
access. This is also intended only for testing, and should be removed
before moving into a production environment.

Remove test database and access to it? [Y/n] Y
- Dropping test database...
... Success!
- Removing privileges on test database...
... Success!

Reloading the privilege tables will ensure that all changes made so far
will take effect immediately.

Reload privilege tables now? [Y/n] Y
... Success!

Cleaning up...

All done! If you've completed all of the above steps, your MySQL
installation should now be secure.

Thanks for using MySQL!
```

sudo chkconfig mysql on(加入到开机启动项中)

第三步：权限授予

```
grant all privileges on *.* to 'root'@'%' identified by 'root';
flush privileges;

mysql> grant all privileges on *.* to 'root'@'%' identified by 'root';
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

OK, 安装成功!

在线安装

卸载已安装的 mysql 依赖

```
rpm -qa | grep mysql 或者 (MySQL)
sudo rpm -e --nodeps `rpm -qa |grep MySQL`
```

```
[user@crxy soft]$ rpm -qa |grep MySQL
MySQL-client-5.1.73-1.glibc23.x86_64
MySQL-server-5.1.73-1.glibc23.x86_64
[user@crxy soft]$ sudo rpm -e --nodeps `rpm -qa |grep MySQL`
[user@crxy soft]$ rpm -qa |grep MySQL
[user@crxy soft]$ █
```

安装

```
sudo yum install -y mysql-server

[user@crxy soft]$ rpm -qa |grep MySQL
[user@crxy soft]$ sudo yum install -y mysql-server
已加载插件: fastestmirror, refresh-packagekit
设置安装进程
Determining fastest mirrors
epel/metalink
  * base: mirror.bit.edu.cn
  * epel: mirrors.neusoft.edu.cn
  * extras: mirrors.btte.net
  * updates: mirrors.opencas.cn
base
epel
epel/primary_db                                         11% [=====
```

初始化设置(同离线安装的方式)

```
sudo service mysqld start
mysql_secure_installation
sudo chkconfig mysqld on(加入到开机启动项中)
```

```
[user@crxy soft]$ sudo service mysqld start
正在启动 mysqld: [确定]
[user@crxy soft]$
[user@crxy soft]$ mysql_secure_installation
```

权限授予、编码设置

第二步：权限授予

```
grant all privileges on *.* to 'root'@'%' identified by 'root';
flush privileges;
```

```
mysql> grant all privileges on *.* to 'root'@'%' identified by 'root';
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

注意：离线安装中的 `mysql` 服务为 `mysql`, 在线安装中 `mysql` 服务为 `mysqld`

离线服务操作 `sudo service mysql [start|stop|restart|status]`

在线服务操作 `sudo service mysqld [start|stop|restart|status]`

1.2.5.2. Hive 的安装

这里的 hadoop 使用的是 hadoop-2.4.1, hive 使用的是 hive-0.14.0。

Hive 安装包下载地址: <http://hive.apache.org/downloads.html>

解压 **Hive** 文件:

进入`$HIVE_HOME/conf/`修改文件

```
cp hive-env.sh.template hive-env.sh
cp hive-default.xml.template hive-site.xml
```

修改`$HIVE_HOME/bin`的 `hive-env.sh`, 增加以下三行

```
export JAVA_HOME=/home/crxy/deployed-soft/jdk1.7.0_55
export HIVE_HOME=/home/crxy/deployed-soft/hive-0.14.0
export HADOOP_HOME=/home/crxy/deployed-soft/hadoop-2.4.1
```

修改`$HIVE_HOME/conf/hive-site.xml`

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

```
<value>jdbc:mysql://localhost:3306/hive?createDatabaseIfNot  
Exist=true</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>root</value>
```

```
</property>
```

新版本(0.12.0以上)中需要增加以下配置

```
<property>
```

```
  <name>hive.querylog.location</name>
```

```
  <value>/home/crxy/deployed-soft/hive-0.14.0/tmp</value>
```

```
</property>
```

```
<property>
```

```
  <name>hive.exec.local.scratchdir</name>
```

```
  <value>/home/crxy/deployed-soft/hive-0.14.0/tmp</value>
```

```
</property>
```

```
<property>
```

```
  <name>hive.downloaded.resources.dir</name>
```

```
  <value>/home/crxy/deployed-soft/hive-0.14.0/tmp</value>
```

```
</property>
```

注意: `hive` 的 `metastore` 数据的编码必须为 `latin1`

```
mysql> show create database hive;
+-----+
| Database | Create Database
+-----+
| hive      | CREATE DATABASE `hive` /*!40100 DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci */ |
+-----+
1 row in set (0.00 sec)          修改数据库编码sql

mysql> alter database hive character set latin1;
Query OK, 1 row affected (0.01 sec)

mysql> show create database hive;
+-----+
| Database | Create Database
+-----+
| hive      | CREATE DATABASE `hive` /*!40100 DEFAULT CHARACTER SET latin1 */ |
+-----+
1 row in set (0.00 sec)          存储引擎的数据库编码必须是latin1
```

不然是会报错的。下图中的错误就是数据库编码不是 latin1 操作的

```
    at org.apache.hadoop.util.RunJar.run(RunJar.java:221)
    at org.apache.hadoop.util.RunJar.main(RunJar.java:136)
NestedThrowableStackTrace:
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Specified key was too long; max key length is 767 bytes
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:57)
    at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
    at java.lang.reflect.Constructor.newInstance(Constructor.java:526)
    at com.mysql.jdbc.Util.handleNewInstance(Util.java:411)
    at com.mysql.jdbc.Util.getInstance(Util.java:386)
    at com.mysql.jdbc.SQLException.createSQLException(SQLException.java:1052)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3597)
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:3529)
```

1.2.5.3. Hive 的运行及访问方式

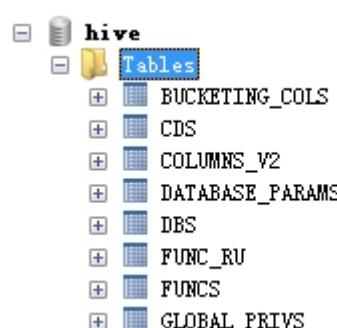
1) CLI 的访问方式

这是一种常见，也是使用最多的访问方式

注意:启动hive之前需要将mysql的java驱动包拷贝到\$HIVE_HOME/lib目录下
在\$HIVE_HOME/bin的目录下执行hive脚本，即可进入hive终端

```
[crxy@master bin]$ ./hive  
Logging initialized using configuration in jar:  
SLF4J: Class path contains multiple SLF4J bindings
```

使用工具连接进入 metastore 存储库，查看元数据信息



至此，hive 安装配置成功

2) 执行 Hive 脚本

有两种方式，在 hive>终端执行，还有在 linux 终端执行

这里的 hql 脚本，比如为 hive.sql，里面的内容为

```
[crxy@master ~]$ more hive.hql
select * from default.t1;
[crxy@master ~]$
```

数据库的内容

```
hive (default)> select * from t1;
OK
1
2
3
4          当前数据库为 default, 查询t1表的结果
5
Time taken: 0.076 seconds, Fetched: 5 row(s)
```

1°、linux 终端

sh \$HIVE_HOME/bin/hive -f hive.hql(的路径)

```
[crxy@master ~]$ sh deployed-soft/hive-0.14.0/bin/hive -f hive.hql
Logging initialized using configuration in file:/home/crxy/deployed-soft/hive-0.14.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/crxy/deployed-soft/hadoop-2.4.1/share/hadoop
inder.class]
SLF4J: Found binding in [jar:file:/home/crxy/deployed-soft/hive-0.14.0/lib/hive-jdb
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
OK
1
2
3
4
5
Time taken: 1.665 seconds, Fetched: 5 row(s)
```

2°、hive 终端

hive> source hive.hql(hql 脚本的 linux 绝对或相对路径)

```
hive> source hive.hql;
OK
1
2
3
4
5
Time taken: 0.458 seconds, Fetched: 5 row(s)
```

3) Java API 代码的方式

hive 远程服务 (端口号 10000) 启动方式

```
#hive --service hiveserver2  
org.apache.hive.jdbc.HiveDriver
```

在 java 代码中调用 hive 的 JDBC 建立连接

4) WebGUI 的方式

这里简单的说一下，WebGUI 的搭建和访问过程

1°、下载 hive 源码包 **apache-hive-0.14.0-src.tar.gz**

2°、解压之某一个目录并进入 hwi 子目录

```
tar -zxvf apache-hive-0.14.0-src.tar.gz -C ... (具体目录)
```

3°、制作 war 包

```
jar cvfM0 hive-hwi-0.14.0.war -C web/ .
```

4°、拷贝 `hive-hwi-0.14.0.war` 至 `$HIVE_HOME/lib` 目录，同时拷贝 `$JAVA_HOME/lib` 下的 `tools.jar` 至 `$HIVE_HOME/lib`

5°、修改 `hive-site.xml` 配置文件

```
<property>  
    <name>hive.hwi.listen.host</name>  
    <value>master.hive.crxy.com</value>  
</property>  
  
<property>  
    <name>hive.hwi.listen.port</name>  
    <value>9999</value>  
</property>  
  
<property>  
    <name>hive.hwi.war.file</name>  
    <value>lib/hive-hwi-0.14.0.war</value>  
</property>
```

5°、启动 hive，及其访问

```
./hive --service hwi >/dev/null 2>&1 &
```

我们可以在浏览器中通过

<http://<ip 地址>:9999/hwi,>

我这里访问为 <http://master.hive.crxy.com:9999/hwi/>, 出现的结果

The screenshot shows two views of the Hive Web Interface (HWI). The top view displays the main interface with a sidebar containing links for Home, Authorize, Browse Schema, Create Session, List Sessions, and Diagnostics. The right panel is titled 'Hive Web Interface' and provides an overview of what HWI offers. The bottom view is a detailed view of a specific table, 't1'. It shows the table name 't1' highlighted with a red box, its ColsSize (1), Input Format (org.apache.hadoop.mapred.TextInputFormat), Output Format (org.apache.hadoop.hive.io.HiveIgnoreKeyTextOutput), Is Compressed? (false), Location (hdfs://master.hive.crxy.com:9000/user/hive/warehouse/t1), and Number Of Buckets (-1). Below this, a 'Field Schema' table is shown with one row: id (Type int, Comment null). A 'Bucket Columns' section is also present.

然并卵~

1. 3. 考题

1. 4. 实验

1. 4. 1. 搭建 **Hive** 开发环境

1. 4. 1. 1. 环境

(1) CentOS6.5

- (2) 带有 1.7 的 jdk
- (3) Hadoop 支持 (2.4 以上)
- (4) Hive0.14.0
- (5) mysql

1.4.1.2. 步骤

- (1) 安装 Mysql。
- (2) 解压 apache-hive-0.14.0-bin.tar.gz 至目录 /home/crxy/deployed-soft/
- (3) 重名为 hive-0.14.0
- (4) 进入 \$HIVE_HOME/conf 目录下面

```
cp hive-env.sh.template hive-env.sh
cp hive-default.xml.template hive-site.xml
```
- (5) 修改 \$HIVE_HOME/bin 的 hive-env.sh, 按照个人情况增加以下三行

```
export JAVA_HOME=/home/crxy/deployed-soft/jdk1.7.0_55
export HIVE_HOME=/home/crxy/deployed-soft/hive-0.14.0
export HADOOP_HOME=/home/crxy/deployed-soft/hadoop-2.4.1
```
- (6) 修改 \$HIVE_HOME/conf/hive-site.xml 按照个人情况修改一下内容

```
<property>
  <name>javax.jdo.option.ConnectionURL</name>
  <value>jdbc:mysql://localhost:3306/hive?createDatabaseIfNot
Exist=true&&characterEncoding=latin1</value>
</property>
<property>
  <name>javax.jdo.option.ConnectionDriverName</name>
  <value>com.mysql.jdbc.Driver</value>
</property>
<property>
  <name>/home/crxy/deployed-soft/hive-0.14.0</name>
```

```
<value>root</value>
</property>
<property>
    <name>javax.jdo.option.ConnectionPassword</name>
    <value>root</value>
</property>
<property>
    <name>hive.querylog.location</name>
    <value>/home/crxy/deployed-soft/hive-0.14.0</value>
</property>
<property>
    <name>hive.exec.local.scratchdir</name>
    <value>/home/crxy/deployed-soft/hive-0.14.0</value>
</property>
<property>
    <name>hive.downloaded.resources.dir</name>
    <value>/home/crxy/deployed-soft/hive-0.14.0</value>
</property>
```

(7) 将 mysql-java-connetor.jar 拷贝到\$HIVE_HOME/lib 目录下

(8) 启动 hive

启动\$HIVE_HOME/bin 下的 hive 脚本

1.4.1.3. 验证

能够进入 hive 客户端，能进行简单的操作，说明 hive 安装配置已成功

```
hive> show databases;
OK
default
Time taken: 1.436 seconds, Fetched: 1 row(s)
hive> use default;
OK
Time taken: 0.22 seconds
hive> create table t1(id int);
OK
Time taken: 0.873 seconds
hive> select * from t1;
OK
Time taken: 1.344 seconds
hive> drop table t1;
OK
```

第 2 次课 Hive 简单操作（一）

2.1. 教学目标

2.1.1. 了解 Hive 日志信息

2.1.2. 掌握数据库相关概念和操作

2.2. 课程内容

2.2.1. Hive 日志信息

2.2.1.1. 去掉 Hive 启动时候的日志警告信息

可能有些同学看着每次启动 Hive 的时候待了一堆日志信息，感觉不爽，就像去掉这些日志信息，那么该如何去掉这些日志信息，让启动变得小清新呢？其实非常的简单：

```
[crxy@master ~]$ sh deployed-soft/hive-0.14.0/bin/hive
Logging initialized using configuration in jar:file:/home/crxy/deployed-soft/hive-0.14.0/lib/hive-common-0.14.0.jar!/hive-log4j.properties
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/crxy/deployed-soft/hadoop-2.4.1/share/hadoop/common/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/crxy/deployed-soft/hive-0.14.0/lib/hive-jdbc-0.14.0-standalone.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]
```

修改\$HIVE_HOME/lib 目录下的 log4j 相关 jar 的文件名后缀为 .bak，让其失效。

```
[crxy@master lib]$ mv hive-jdbc-0.14.0-standalone.jar hive-jdbc-0.14.0-standalone.jar.bak
[crxy@master lib]$ pwd
/home/crxy/deployed-soft/hive-0.14.0/lib
[crxy@master lib]$
```

====>小清新了~

```
[crxy@master bin]$ ./hive
Logging initialized using configuration in file:/home/crxy/deployed-soft/hive-0.14.0/conf/hive-log4j.properties
hive>
```

2.2.1.2. Hive 的日志

当我们遇到 Hive 错误的时候，我们要学会去查看 Hive 的日志信息，通过日志的提示

来分析提示，找到错误的根源，帮助我们及时解决错误。

那么我在在哪里查看 Hive 日志呢，我们可以通过配置文件来看。

\$HIVE_HOME/conf 目录下有 `hive-log4j.properties.template` 文件，我们

```
[crxy@master conf]$ ll
total 308
-rw-r--r--. 1 crxy crxy 1139 Nov  8 2014 beeline-log4j.properties.template
-rw-r--r--. 1 crxy crxy 144936 Nov  8 2014 hive-default.xml.template
-rw-r--r--. 1 crxy crxy 2542 Dec 24 01:49 hive-env.sh
-rw-r--r--. 1 crxy crxy 2378 Nov  8 2014 hive-env.sh.template
-rw-r--r--. 1 crxy crxy 2662 Nov  8 2014 hive-exec-log4j.properties.template
-rw-r--r--. 1 crxy crxy 3050 Nov  8 2014 hive-log4j.properties.template
-rw-r--r--. 1 crxy crxy 144914 Dec 24 01:57 hive-site.xml
```

将其修改为 `hive-log4j.properties`，查看文件内容我们发现，

<code>hive.log.threshold=ALL</code>	Hive 日志存放目录
<code>hive.root.logger=INFO, DRFA</code>	
<code>hive.log.dir=\${java.io.tmpdir}/\${user.name}</code>	hive.log.dir 存放
<code>hive.log.file=hive.log</code>	

在`${java.io.tmpdir}/${user.name}`，我们可以通过一个简单的 jdk 中的 System



SystemInfo.java

类来获取`${java.io.tmpdir}`、`${user.name}`。

```
[crxy@master data]$ java SystemInfo | grep java.io.tmpdir
java.io.tmpdir=/tmp
[crxy@master data]$ java SystemInfo | grep user.name
user.name=crxy
[crxy@master data]$
```

那么我们就知道了 Hive 日志的目录在`/tmp/crxy/`，文件名称为 `hive.log`，以后查看日志就方便多了。

```
[crxy@master data]$ ll /tmp/crxy/
total 536
-rw-rw-r--. 1 crxy crxy 547649 Dec 24 05:20 hive.log
[crxy@master data]$
```

当然，我们也可以按照我们的要求来个性化修改。

`hive-exec-log4j.properties.template` 可以做同样的操作，作为我们监控 `hive` 的执行日志。

2.2.2. Hive 中的数据库、表、数据与 HDFS 的对应关系

2.2.2.1. Hive 数据库

我们通过在 hive 终端，查看数据库信息，可以看出来 hive 有一个默认的数据库 default，而且我们也知道，hive 数据库对应的是 hdfs 上面的一个目录，那么默认的 default 数据库到底是对应的哪一个目录呢？我们可以通过 hive 配置文件 (hive-site.xml) 中一个 hive.metastore.warehouse.dir 配置项看到信息。

```
<property>
  <name>hive.metastore.warehouse.dir</name>
  <value>/user/hive/warehouse</value>
  <description>location of default database for the warehouse</description>
</property>
```

如上图，它告诉了我们默认数据库 default 在 hdfs 的目录。

Browse Directory

Permission	Owner	Group	Size	Replication	Block Size	Name
014.						

014.

在我们的 metastore 中，我们可以查阅表 DBS 来获知对应关系。

DB_ID	DESC	DB_LOCATION_URI	NAME	OWNER_NAME	OWNER_TYPE
1	Default Hive database	hdfs://master.hive.crxy.com:9000/user/hive/warehouse	default	public	ROLE

2.2.2.1.1. 查看并使用数据库

查看使用命令: show databases;

```
hive> show databases;
OK
default
Time taken: 0.083 seconds, Fetched: 1 row(s)
hive>
```

选择数据库: use dbName;

```
eg: use default;
```

```
hive> use default;
OK
Time taken: 0.072 seconds
hive> [REDACTED]
```

2.2.2.1.2. 创建数据库

类似于 mysql，在这里创建两个数据为例，一个创建数据库，我们不指定位置，第二个制定它在 hdfs 上面的具体目录。

eg.

不指定位置：

```
create database mydb1;
```

```
hive> create database mydb1;
OK
Time taken: 0.616 seconds
hive> show databases;
OK
default
mydb1
Time taken: 0.07 seconds, Fetched: 2 row(s)
hive> [REDACTED]
```

查看 HDFS 目录，那么也就是说在 /user/hive/warehouse 目录下面。

Browse Directory

/user/hive/warehouse							Go!
Permission	Owner	Group	Size	Replication	Block Size	Name	
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	mydb1.db	

一旦创建成功了，metastore 就感知到了，更新相应的表内容

DB_ID	DESC	DB_LOCATION_URI	NAME	OWNER_NAME	OWNER_ROLE
1	Default Hive database	hdfs://master.hive.crxy.com:9000/user/hive/warehouse	default	public	ROLE
6	(NULL)	hdfs://master.hive.crxy.com:9000/user/hive/warehouse/mydb1.db	mydb1	crxy	USER
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

创建数据库制定位置

```

create database mydb2 location '/user/hive/mydb2';
Time taken: 0.07 seconds, Fetched: 2 row(s)
hive> create database mydb2 location '/user/hive/mydb2';
OK
Time taken: 0.228 seconds
hive>

```

/user/hive Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	mydb2
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	warehouse

查看 hdfs 目录，可以看到 mydb2 和 warehouse 是平级目录，这在我们的认知上，更符合数据库与数据库之间是平级关系。metastore 中也做了相应的更新。

DB_ID	DESC	DB_LOCATION_URI	NAME
1	Default Hive database	hdfs://master.hive.crxy.com:9000/user/hive/warehouse	default
6	(NULL)	hdfs://master.hive.crxy.com:9000/user/hive/warehouse/mydb1.db	mydb1
7	(NULL)	hdfs://master.hive.crxy.com:9000/user/hive/mydb2	mydb2
*	(NULL)	(NULL)	(NULL)

2.2.2.1.3. 删除数据库

非常简单，使用 drop 命令，和 mysql 一样，

```
drop database <dbName>.
```

eg.

```
drop database mydb1;
```

```
drop database mydb2;
```

```

hive> drop database mydb1;
OK
Time taken: 0.569 seconds
hive> drop database mydb2;
OK
Time taken: 0.245 seconds
hive> show databases;
OK
default
Time taken: 0.091 seconds, Fetched: 1 row(s)
hive>

```

元数据中及时去掉了相关信息。

DB_ID	DESC	DB_LOCATION_URI	NAME
1	Default Hive database	hdfs://master.hive.crxy.com:9000/user/hive/warehouse	default

注意：默认数据库是无法删除的!!!

```
hive> drop database default;
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. MetaException(message:Can not drop default database)
hive>
```

2.2.2.2. Hive 数据类型

hive 呢，作为一个类似 db 的东东，也有自己的数据类型，便于存储、统计、分析。

数据类型	所占字节	开始支持版本
TINYINT	1byte, -128 ~ 127	
SMALLINT	2byte, -32,768 ~ 32,767	
INT	4byte,-2,147,483,648 ~ 2,147,483,647	
BIGINT	8byte,-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807	
BOOLEAN		
FLOAT	4byte单精度	
DOUBLE	8byte双精度	
STRING		
BINARY		从Hive0.8.0开始支持
TIMESTAMP		从Hive0.8.0开始支持
DECIMAL		从Hive0.11.0开始支持
CHAR		从Hive0.13.0开始支持
VARCHAR		从Hive0.12.0开始支持
DATE		从Hive0.12.0开始支持

复杂类型包括ARRAY,MAP,STRUCT,UNION，这些复杂类型是由基础类型组成的。

在下面的使用过程中呢，会着重说明一下 date、boolean 和复杂的数据类型，其余的数据类型和 mysql 中的几乎一样，就不做过多说明了。

知道了 Hive 有哪些数据类型了，那么我们就可以来进行操作了，操作 DDL、DQL 以及 DML 了，下面我们来看一下 Hive 的表。

2.2.2.3. Hive 表

同数据库一样，hive 中的表，对应到 hdfs 是文件夹。

2.2.2.3.1. 创建表

语句非常简单，举简单的例子，`create table t1(id int);`

```
hive> create table t1(id int);;
OK
Time taken: 1.193 seconds
hive> show databases.
```

2.2.2.3.2. 查看所有的表

`show tables;`

```
hive> show tables;
OK
t1
Time taken: 0.108 seconds, Fetched: 1 row(s)
```

2.2.2.3.3. 查看表结构

`desc tableName;` eg. `desc t1;`

```
hive> desc t1;
OK
id                  int
Time taken: 1.007 seconds, Fetched: 1 row(s)
```

查看表的创建信息

```

hive> show create table t1;
OK
CREATE TABLE `t1`(
  `id` int)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.mapred.TextInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
LOCATION
  'hdfs://master.hive.crxy.com:9000/user/hive/warehouse/t1'
TBLPROPERTIES (
  'COLUMN_STATS_ACCURATE'='false',
  'last_modified_by'='crxy',
  'last_modified_time'='1451122574',
  'numFiles'='3',
  ' numRows'='-1',
  ' rawDataSize'='-1',
  ' totalSize'='95',
  ' transient_lastDdlTime'='1451122574')
Time taken: 0.168 seconds, Fetched: 19 row(s)

```

2.2.2.3.4. 表在 hdfs 中的位置

/user/hive/warehouse							Gc
Permission	Owner	Group	Size	Replication	Block Size	Name	
dwxr-xr-x 是一个文件夹	crxy	supergroup	0 B	0	0 B	t1	

2.2.2.3.5. 表在元数据 metastore 中的体现

这里可以看到DB_ID对应的是默认数据库default的在metastore中的id，也看到TBL_NAME是我们创建的t1

TBL_ID	CREATE_TIME	DB_ID	LAST_ACCESS_TIME	OWNER	RETENTION	SD_ID	TBL_NAME	TBL_TYPE	
(NULL)	1450972443	11	(NULL)	0	crxy	0	11	t1	MANAGED_TABLE

在表COLUMNS_V2中存储的hive表的字段信息

那表与字段的关联关系是通过表 SDS 来维护的

CD_ID指向的我们刚刚查看的表COLUMNS_V2中的主键和外键CD_ID

location指向的是我们刚刚创建的表t1

2.2.2.3.6. 加载数据到表中的两种方式

1) 使用命令 load data

以 t1 为例，在我本机/home/crxy/data/下有一个 t1 文件，将其加载到 t1 表中

```
[crxy@master data]$ pwd
/home/crxy/data
[crxy@master data]$ more t1
1
2
3
4
5
```

```
load data local inpath '/home/crxy/data/t1' into table t1;
```

```
hive> load data local inpath '/home/crxy/data/t1' into table t1;
Loading data to table default.t1
Table default.t1 stats: [numFiles=1, totalSize=10]
OK
Time taken: 3.538 seconds
```

查看表中的内容

```

hive> select * from t1;
OK
1
2
3
4
5
Time taken: 1.243 seconds, Fetched: 5 row(s)

```

2) hadoop shell

可以使用 hadoop fs -put 命令直接将数据放到表在 hdfs 的目录下面

eg.hdfs dfs -put t1 /user/hive/warehouse/t1/t1_1

```

[crxy@master data]$ hdfs dfs -put t1 /user/hive/warehouse/t1/t1_1
[crxy@master data]$

```

/user/hive/warehouse/t1

Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	crxy	supergroup	10 B	1	128 MB	t1
-rw-r--r--	crxy	supergroup	10 B	1	128 MB	t1_1

查看表内容，我们得到了同样的效果

```

hive> select * from t1;
OK
1
2
3
4
5
1
2
3
4
5
Time taken: 0.192 seconds, Fetched: 10 row(s)

```

2. 2. 2. 3. 7. 数据加载的两种模式

一般的，我们在加载数据到数据库有两种模式，一种是读模式，一种是写的模式。

写模式指的是在数据加载的时候，对数据合法性进行校验。表中的数据是合乎规范的，有利于快速查询和快速处理。

读模式指的是在数据读取的时候，对数据合法性进行校验，在写入的时候不校验。优点是加载数据非常快，适合于大量数据的一次性加载。

Hive 是读模式。当上传文件到表中, hive 不会对表中的数据进行校验, 对于错误的数据格式查询的时候也不会报错, 只是显示为 **NULL**。

eg. 如果我们上传一个不同于 t1 建表结构的文件 t2

```
[crxy@master data]$ more t2
1      zhangsan      2014-01-13      true
2      李四        2015-11-11      false
3      王五        2014-23-23      0

Time taken: 0.001 seconds, Fetched: 3 row(s)
hive> load data local inpath '/home/crxy/data/t2' into table t1;
Loading data to table default.t1
Table default.t1 stats: [numFiles=3, totalSize=95]
OK
Time taken: 1.085 seconds
hive> select * from t1;
OK
1
2
3
4
5
1
2  hive无法识别, 不同于自身表结构的文件, 在加载load的时候不会报错,
3  只会在查询显示的时候, 显示为NULL
4
5
NULL
NULL
NULL
```

2.2.2.4. 修改表, 表的常用 DDL

2.2.2.4.1. 重命名表名

```
ALTER TABLE tblName RENAME TO new_tblName

hive> alter table t1 rename to t1_bak;
OK
Time taken: 0.806 seconds
hive> show tables;
OK
t1_bak
```

2.2.2.4.2. 给表增加一个字段:

使用: **alter table tblName add columns (columnName type comment 'info');**

eg.

```

hive> alter table t1 add columns (name string);
OK
Time taken: 0.87 seconds
hive> desc t1;
OK
id          int
name        string
Time taken: 0.472 seconds, Fetched: 2 row(s)
hive> select * from t1;
OK
1      NULL
2      NULL
3      NULL

```

2.2.2.4.3. 修改某一个字段:

```

ALTER TABLE table_name CHANGE
[COLUMN] col_old_name col_new_name column_type
[COMMENT col_comment]
[FIRST|AFTER column_name];

```

eg.

```
ALTER TABLE tblName CHANGE colName new_colName new_type
```

```

> alter table t1 change name stu_name string comment "stu's name";
OK
Time taken: 1.065 seconds
hive> desc t1;
OK
id          int
stu_name    string          stu's name
Time taken: 0.879 seconds, Fetched: 2 row(s)

```

修改字段在表定义中的位置，原来的数据

```

hive> alter table t1 add columns(age int, gender string);
OK
Time taken: 1.034 seconds
hive> desc t1;
OK
id          int
stu_name    string          stu's name
age         int
gender     string
Time taken: 0.454 seconds, Fetched: 4 row(s)

```

将 age 放到第一列

```

TIME taken: 0.151 seconds, Fetched: 1 row(s)
hive> alter table t1 change age age int first;
OK
Time taken: 0.742 seconds
hive> desc t1;
OK
age          int
id           int
stu_name      string
gender        string
stu's name
Time taken: 0.451 seconds, Fetched: 4 row(s)

```

将 age 放在某一列的后面

```

hive> alter table t1 change age age int after id;
OK
Time taken: 0.976 seconds
hive> desc t1;
OK
id           int
age          int
stu_name      string
gender        string
stu's name
Time taken: 0.38 seconds, Fetched: 4 row(s)

```

2.2.2.4.4. Replace 替换:

`ALTER TABLE tblName REPLACE COLUMNS (col_spec[, col_spec ...])`

```

hive> alter table t1_bak replace columns (
    > t_id int comment 't_id'
    > );
OK
Time taken: 0.653 seconds
hive> desc t1_bak;
OK
t_id          int
t_id
Time taken: 0.371 seconds, Fetched: 1 row(s)
hive> select * from t1_bak;
OK
1
2

```

这有点删除表中某一列的意思。

2.2.2.5. 列分隔符

在我们工作的过程中呢，肯定不会简简单单的向上述的创建一张非常简单的表，就相对复杂的多了，这里我们创建一张相对复杂的表：

```
create table t2 (
```

```
    id int comment 't2's ID',
    stu_name string comment 'name',
    stu_birthday date comment 'birthday',
    online boolean comment 'is online'
);
```

这样创建没有问题，我们来看一下，加载数据，看一下表中的数据，会不会有问题呢？

```
hive> create table t2(
> id int comment 'ID',
> stu_name string comment 'name',
> stu_birthday date comment 'birthday',
> online boolean comment 'is online'
> );
OK
Time taken: 0.735 seconds

hive> load data local inpath '/home/crxy/data/t2' into table t2;
Loading data to table default.t2
Table default.t2 stats: [numFiles=1, totalSize=75]
OK
Time taken: 2.639 seconds
hive> select * from t2;
OK
NULL      NULL      NULL      NULL
NULL      NULL      NULL      NULL
NULL      NULL      NULL      NULL
Time taken: 1.502 seconds, Fetched: 3 row(s)
hive>
```

唉，我们发现不是我们想要的结果，数据没有被识别，这是为什么？

OK，hive 在创建表的时候呢，告诉我们需要指定相应的行分隔符，列分隔符，而我们在创建 mysql 表的时候，这些都是不需要的，因为它在组织数据的时候，已经规定死了数据的表现形式。

那么，我们刚才在创建 t2 的时候没有指定相应的分隔符，hive 应该是有相应的分隔符的，不然 t1 也看不到相应的数据了。

实际上呢，**hive 是有默认的分隔符的，默认的行分隔符这个大家应该可以看到，比较清楚，是'\n'**，而默认的列分隔符呢，是\001，复杂数据类型的分隔符一会再给大家说明。

\001 这个是 ASCII 码中一些**特殊不常使用不可见字符**，在文本中我们可以通过 ctrl+v 然后 ctrl+a 来输入\001，这里我们在将 t2 改一下，重新上传，在查看表 t2。

```

1 zhangsan 2014-01-13 true
2 李四 2015-11-11 false
3 王五 2014-23-23 0

hive> load data local inpath '/home/crxy/data/t2_2' into table t2;
Loading data to table default.t2
Table default.t2 stats: [numFiles=2, totalSize=150]
OK
Time taken: 1.291 seconds
hive> select * from t2;
OK
NULL NULL NULL NULL 按照分隔符来读取/分析数据了!
NULL NULL NULL NULL
NULL NULL NULL NULL
1 zhangsan 2014-01-13 true
NULL NULL NULL NULL
NULL NULL NULL NULL
Time taken: 0.229 seconds, Fetched: 6 row(s)

```

那么问题来了，为了我们能够在上传数据之后就能看到相应的数据，那我们该如何设置 hive 表的默认分隔符呢？

其实也是非常简单的，只要我们自创建表的时候指定一下分隔符就可以了，我们把 t2 的 DDL 修改一下，加一些约束分割信息就可以了。

```

create table t2_2(
    id int comment 'ID',
    stu_name string comment 'name',
    stu_birthday date comment 'birthday',
    online boolean comment 'is online'
) row format delimited
    fields terminated by '\t'

```

```

lines terminated by '\n';

hive> create table t2_2 (
    > id int comment 'ID',
    > stu_name string comment 'name',
    > stu_birthday date comment 'birthday',
    > online boolean comment 'is online'
    > ) row format delimited 制定分隔符
    > fields terminated by '\t'
    > lines terminated by '\n';
OK
Time taken: 0.249 seconds
hive> load data local inpath '/home/crxy/data/t2' into table t2_2;
Loading data to table default.t2_2
Table default.t2_2 stats: [numFiles=1, totalSize=75]
OK
Time taken: 1.426 seconds
hive> select * from t2_2;
OK
1 zhangsan 2014-01-13 true 校验数据是无法识别异常
2 李四 2015-11-11 false 数据，只能显示为NULL
3 王五 NULL NULL
Time taken: 0.293 seconds, Fetched: 3 row(s)

```

有了这些知识呢，那我们以后创建表的时候就要想对复杂一点了，也要有一定的格式了。

后面再说负载数据类型的时候呢，再来说一下复杂数据元素之间的默认分隔符。

2.2.2.6. 复杂数据类型

有了上述的知识经验之后呢，我们再来看一下 hive 中该如何使用复杂的数据类型呢？

2.2.2.6.1. array

这里举一个学生有多个爱好的例子，有两个学生，zhangsan、lisi，爱好分别为 swing、sing、coding，poetry、music、football、pingpong，那么我们就可以将爱好看成一个爱好的数组来存放学生的爱好数据，怎么建表呢？

```
create table t3_stu_hobby(
    id int comment 'ID',
    stu_name string comment 'stu name',
    stu_hobby array<string> comment "stu's hobby"
) row format delimited
fields terminated by '\t'
collection items terminated by ',';
```

```
hive> create table t3_stu_hobby (
>   id int comment 'ID',
>   stu_name string comment 'stu name',
>   stu_hobby array<string> comment "stu's hobby"
> ) row format delimited
> fields terminated by '\t'
> collection items terminated by ',';
OK
Time taken: 4.525 seconds
```

数据：

```
[crxy@master data]$ more t3_stu_hobby
1      zhangsan      swing,sing,coding
2      lisi      poetry,music,football,pingpong
```

加载和查询：

```

hive> load data local inpath '/home/crxy/data/t3_stu_hobby' into table t3_stu_hobby;
Loading data to table default.t3_stu_hobby
Table default.t3_stu_hobby stats: [numFiles=1, totalSize=67]
OK
Time taken: 6.345 seconds
hive> select * from t3_stu_hobby;
OK
1      zhangsan      ["swing","sing","coding"]
2      lisi          ["poetry","music","football","pingpong"]
Time taken: 1.798 seconds, Fetched: 2 row(s)
hive> 

```

查询数组中的某一个元素使用 `arrayName[index]`, 如下图, 可以和上图数据对应

```

hive> select id, stu_name, stu_hobby[1] from t3_stu_hobby;
OK
1      zhangsan      sing
2      lisi          music
Time taken: 0.721 seconds, Fetched: 2 row(s)
hive> 

```

实际上呢, 说明一下 hive 元素之间的默认分隔符为`\002`, 在键盘上通过 **ctrl+v** 和 **ctrl+b** 来输入。

2.2.2.6.2. map

上面说了 array 数组之后呢, 来说一下另外一种常见的结合—map, 我们知道 map 集合里面存储的是键值对, 那么每一个键值对之间是集合的一个 item, 默认分隔符我们知道, 是`\002`, 那键 key 和值 value 之间又用什么来分割呢? 好了, 不卖关子了, 实际上我们顺着上面的`\001` 和`\002` 往后排就是了, 分隔符为`\003`, 怎么敲呢, **ctrl+v** 和 **ctrl+c**, 这里给大家举个例子, 说有两个学生 zhangsan、lisi, 每个学生考了试, 有语文、数学、英语分别对应为 60、61、62, 91、60、32。

```

[crxy@master data]$ vi t4_stu_scores
1      zhangsan      chinese:60,math:61,english:62
2      lisi          chinese:90,math:60,english:32
~ 

```

建表语句

```

create table t4_stu_scores(
    id int comment 'ID',
    stu_name string comment 'stu name',
    stu_scores map<string, int> comment "stu's scores"
) row format delimited

```

```
fields terminated by '\t'  
collection items terminated by ','  
map keys terminated by ':';
```

```
hive> create table t4_stu_scores (  
> id int comment 'ID',  
> stu_name string comment "stu's name",  
> stu_scores map<string, int> comment "stu's scores"  
> ) row format delimited  
> fields terminated by '\t'  
> collection items terminated by ','  
> map keys terminated by ':';  
OK  
Time taken: 0.645 seconds  
hive> █
```

如何查阅 map 中的数据呢？使用 `mapName['fieldName']`，eg.

```
hive> load data local inpath '/home/crxy/data/t4_stu_scores' into table t4_stu_scores;  
Loading data to table default.t4_stu_scores  
Table default.t4_stu_scores stats: [numFiles=1, totalSize=78]  
OK  
Time taken: 2.01 seconds  
hive> select * from t4_stu_scores;  
OK  
1      zhangsan      {"chinese":60,"math":61,"english":62}  
2      lisi          {"chinese":90,"math":60,"english":32}  
Time taken: 0.722 seconds, Fetched: 2 row(s)  
hive> select id, stu_name, stu_scores['chinese'], stu_scores['math'] from t4_stu_scores;  
OK  
1      zhangsan      60      61  
2      lisi          90      60  
Time taken: 0.496 seconds, Fetched: 2 row(s)
```

这里再说明一下默认的 **k-v** 分隔符

建表语句：

```
create table t4(  
    id int comment 'ID',  
    stu_name string,  
    stu_scores map<string, int>  
) row format delimited  
    fields terminated by '\t'  
    collection items terminated by ',';
```

```

hive> create table t4(
    > id int,
    > name string,
    > scores map<string, int>
    > ) row format delimited
    > fields terminated by '\t'
    > collection items terminated by ',';
OK
Time taken: 0.491 seconds
hive> █

```

数据:

```

[crxy@master data]$ more t4
1      zhangsan      chinese:60,math:61,english:62
2      lisi      chinese:90,math:60,english:32
[crxy@master data]█

```

加载并查询:

```

hive> load data local inpath '/home/crxy/data/t4' into table t4;
Loading data to table default.t4
Table default.t4 stats: [numFiles=1, totalSize=78]
OK
Time taken: 2.722 seconds
hive> select * from t4;
OK
1      zhangsan      {"chinese":60,"math":61,"english":62}
2      lisi      {"chinese:90":null,"math:60":null,"english:32":null}
Time taken: 0.548 seconds, Fetched: 2 row(s)
hive> █

```

第一行的数据被识别出来了，第二列的数据没有识别。

2.2.2.6.3. struct

再来介绍最后一种复杂类型 struct，有点像我们 java 中的对象，举个例子说明一下，超人学院有 2 个员工，zhangsan、lisi，每个员工都有地址信息，一个是家里的地址，一个是单位的地址，我们来组织数据，建一张表 t5_staff_address

```

create table t5_staff_address (
    id int comment 'ID',
    staff_name string comment 'staff name',
    staff_address struct<home_addr:string, office_addr:string>
    comment "staff's address",
) row format delimited
fields terminated by '\t'

```

```

collection items terminated by ',',';

Time taken: 0.000 seconds
hive> create table t5_staff_address (
  > id int comment 'ID',
  > staff_name string comment "staff's name",
  > staff_address struct<home_addr:string, office_addr:string>
  > )row format delimited
  > fields terminated by '\t'
  > collection items terminated by ',';    元素之间的分隔符,用collection items
OK
Time taken: 0.424 seconds
hive> load data local inpath '/home/crxy/data/t5_staff_address' into table t5_staff_address;
Loading data to table default.t5_staff_address
Table default.t5_staff_address stats: [numFiles=1, totalSize=54]
OK
Time taken: 1.523 seconds
hive> select * from t5_staff_address;
OK
1      zhangsan      {"home_addr":"gansu","office_addr":"lishuiqiao"}
2      lisi          {"home_addr":"haidian","office_addr":"lishuiqiao"}
Time taken: 0.442 seconds, Fetched: 2 row(s)
hive> select id, staff_name, staff_address.home_addr from t5_staff_address;
OK
1      zhangsan      gansu                      数据引用有点像java中的对象属性引用
2      lisi          haidian
Time taken: 0.392 seconds, Fetched: 2 row(s)
hive> 

```

2.2.2.6.4. 综合案例

超人学院有员工，有员工表 employees

```

create table employees (
  id int,
  salary float,
  subordinates array<string>,
  deductions map<string, float>,
  address struct<city:string, province:string, zip:int>
) row format delimited
fields terminated by '\001'
collection items terminated by '\002'
map keys terminated by '\003'
lines terminated by '\n';

hive> create table employees(
  > id int,
  > name string,
  > salary float,
  > subordinates array<string>,
  > deductions map<string, float>,
  > address struct<city:string, province:string, zip:int>
  > ) row format delimited
  > fields terminated by '\001'
  > collection items terminated by '\002'
  > map keys terminated by '\003'
  > lines terminated by '\n';           使用默认的分隔符
OK
Time taken: 0.086 seconds

```

数据

```
[crxy@master data]$ vi employees
1 zhangsan 3000.0 {"social security":0.05,"individual income tax":0.03,"shenzhen","guangdong",523000}
2 lisi 4000.0 {"wangwu","zhangsan","social security":0.05,"individual income tax":0.03,"haidian","beijing",100000}
3 wangwu 5000.0 {"zhangsan","social security":0.05,"individual income tax":0.03,"shenzhen","changping",100000}
4 zhaoliu 6000.0 {"zhangsan","lisi","wangwu","social security":0.05,"individual income tax":0.03,"lanzhou","gansu",730000}
```

查询

```
hive> load data local inpath 'data/employees' into table employees;
Loading data to table default.employees
Table default.employees stats: [numFiles=1, totalSize=400]
OK
Time taken: 0.341 seconds
hive> select * from employees;
OK
1      zhangsan      3000.0  []      {"social security":0.05,"individual income tax":0.03}  {"city":"shenzhen","province":"guangdong","zip":523000}
2      lisi          4000.0  ["wangwu","zhangsan"]  {"social security":0.05,"individual income tax":0.03}  {"city":"haidian","province":"beijing","zip":100000}
3      wangwu        5000.0  ["zhangsan"]  {"social security":0.05,"individual income tax":0.03}  {"city":"shenzhen","province":"changping","zip":100000}
4      zhaoliu        6000.0  ["zhangsan","lisi","wangwu"]  {"social security":0.05,"individual income tax":0.03}  {"city":"lanzhou","province":"gansu","zip":730000}
Time taken: 0.065 seconds, Fetched: 4 row(s)
hive>
```

2.3. 考题

问：如果在 mysql 中有一张表 student(id, name)，还有一张表 address(stu_id, home, school)，还有联系方式表 contact(stu_id, mine, parents, others)。如果把这三张表迁移到 hive 中，如何迁移哪？

答：

可以一一对应，优点是迁移成本非常低，包括 DDL 和业务逻辑，几乎不需要修改，可以直接使用。缺点是产生大量的表连接，造成查询慢。

可以一对多，mysql 中的多张关联表可以创建为 hive 中的一张表。优点是减少表连接操作。缺点是迁移成本高，需要修改原有的业务逻辑。

实际上，在我们日常的开发过程中遇到这样的问题，要想比较完美、顺利的解决，一般都分为两个阶段，第一个阶段，现在快捷迁移，就是上面说的一一对应，让我们的系统能跑起来，在此基础之上呢，再做去重处理，就是做一张大表，尽量包含以上所有字段，eg

```
stu(id, name, address struct<home,school>, contact struct<...>);
```

等第二个阶段完工之后了呢，就可以去掉跑在新的系统里面了。

2.4. 实验

在 hive 中联系创建数据库、表，以及联系复杂数据类型的建表语句，数据的加载和获

取。

第 3 次课 **Hive** 操作（二）

3.1. 教学目标

3.1.1. 掌握 **Hive** 表的类型

3.1.2. 掌握数据加载语句

3.1.3. 掌握数据的导出

3.2. 课程内容

3.2.1. **Hive** 表的类型

Hive 的表有哪些类型呢，我们简单可以分为四种，受控表、外部表、分区表、桶表，从严格意义上说，应该分为两种受控表，又叫内部表、外部表，分区表和桶表其实是受控表的不同体现。

3.2.1.1. 受控表

所谓受控表，就是说数据的生命周期受表的控制，当表删除的时候，其数据文件一并被删除。

实际上，我们前面创建的表都属于受控表，前面我们已经演示了，创建一张表，其对应的在 `hive` 中就有了记录，在 `metastore` 中有了相应的表定义，当我们一旦从 `hive` 中删除一张表的表定义之后，其表中的数据也不复存在了，在 `metastore` 中定义也不存在了。

3.2.1.2. 外部表

表的定义和数据的生命周期互相不约束，数据只是表对 `hdfs` 上的某一个目录的引用而已，当删除表定义的时候，其表中的数据依然是存在的。

我们这里创建一张外部表，看一下外部表的 DDL 该如何来写。

```
create <external> table tblName (字段 类型...) location hdfs_uri;
```

这里在写外部表的 DDL 的时候需要特别指明是外部表，用到了关键字 external，我们知道，外部表只是对 hdfs 中某一个目录的引用，所以还需要我们指明引用的具体的目录。

eg. 在 hdfs /data/ 下有一个文件 t2，内容如下

```
[crxy@master data]$ hadoop fs -text /data/t2
1      zhangsan      2014-01-13      true
2      李四        2015-11-11      false
3      王五        2014-23-23      0
```

我们这里创建一个外部表 t6_external 来引用一下数据 /data/t2

```
create external table t6_external (
    id int,
    name string,
    birthday date,
    online boolean )
row format delimited fields terminated by '\t' location '/data/';
```

```
hive> create external table t6_external (
    > id int,
    > name string,
    > birthday date,
    > online boolean
    > ) row format delimited
    > fields terminated by '\t'
    > location '/data/';
OK
Time taken: 2.524 seconds
```

它查询数据还是有的，如下图。

```
hive> select * from t6_external;
OK
1      zhangsan      2014-01-13      true
2      李四        2015-11-11      false
3      王五        NULL        NULL
Time taken: 2.196 seconds, Fetched: 3 row(s)
hive> █
```

但是我们浏览一下 hdfs，到 /user/hive/warehouse/ 看一下，是否有相应的表了呢？

居然没有，如下图

/user/hive/warehouse

Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t1
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t2
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t2_2
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t3_stu_hobby
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t4
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t4_stu_scores
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t5_staff_address

在我们来看一下 metastore，这里看到 t6_external 的类型是外部表。

	DB_ID	LAST_ACCESS_TIME	OWNER	RETENTION	SD_ID	TBL_NAME	TBL_TYPE	VIEW_EXP
	1	0	crxy	0	11	t1	MANAGED_TABLE	(NULL)
	1	0	crxy	0	21	t2	MANAGED_TABLE	(NULL)
	1	0	crxy	0	22	t2_2	MANAGED_TABLE	(NULL)
	1	0	crxy	0	26	t3_stu_hobby	MANAGED_TABLE	(NULL)
	1	0	crxy	0	27	t4_stu_scores	MANAGED_TABLE	(NULL)
	1	0	crxy	0	30	t4	MANAGED_TABLE	(NULL)
	1	0	crxy	0	32	t5_staff_address	MANAGED_TABLE	(NULL)
*	1	0	crxy	0	36	t6_external	EXTERNAL_TABLE	(NULL)
*	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

我们删除 t6_external 表定义看看数据是否被删除了呢，按照前面的理论，应该不会被删除的。

```
hive> drop table t6_external;
OK
Time taken: 3.098 seconds
hive> show tables;
OK
t1
t2
t2_2
t3_stu_hobby
t4
t4_stu_scores
t5_staff_address
Time taken: 0.18 seconds, Fetched: 7 row(s)
```

数据还在，如下图：

/data

Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	crxy	supergroup	75 B	1	128 MB	t2

metastore 中的相应的记录也应该一并被删除掉了。

3.2.1.3. 分区表

假设服务器集群每天都产生一个日志数据文件，把数据文件统一存储到 HDFS 中。我们如果想查询某一天的数据的话，hive 执行的时候会对所有文件都扫描一遍，判断是否是指定的日期。

可以让日期作为一个子目录。当 hive 查询的时候，根据日期去判断子目录。然后扫描符合条件的子目录中的数据文件。

3.2.1.3.1. 分区表的 CRUD

按照上面的叙述，我们来创建一个简单的分区表，

```
create table t7_partition_1 (
    id int comment 'ID',
    name string comment 'name'
) partitioned by (create_time date comment 'create time')
row format delimited
fields terminated by '\t';
```

```
hive> create table t7_partition_1 (
    > id int comment 'ID',
    > name string comment 'name'
    > ) partitioned by (create_time date comment 'create time')
    > row format delimited
    > fields terminated by '\t';
OK
Time taken: 0.298 seconds
hive> desc t7_partition_1;
OK
id                      int                      ID
name                     string                    name
create_time              date                     create time
# Partition Information → 多了分区的信息
# col_name                data_type               comment
create_time              date                     create time
Time taken: 0.574 seconds, Fetched: 8 row(s)
```

加载数据：

```
[crxy@master data]$ more t7_partition_1_0
1      zhangsan
2      lisi
[crxy@master data]$
```

```
hive> load data local inpath '/home/crxy/data/t7_partition_1_0' into table t7_partition_1 partition (create_time='2015-12-26');
Loading data to table default.t7_partition_1 partition (create_time=2015-12-26)
Partition default.t7_partition_1{create_time=2015-12-26} stats: [numFiles=1, numRows=0, totalSize=18, rawDataSize=0]
OK
Time taken: 3.466 seconds
```

可以看到的是自动创建了分区，比较智能，来查看一下 hdfs

/user/hive/warehouse/t7_partition_1						
Permission	Owner	Group	Size	Replication	Block Size	Name
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	create_time=2015-12-26
刚才创建的分区在hdfs上面是一个文件夹/目录						
/user/hive/warehouse/t7_partition_1/create_time=2015-12-26						
Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	crxy	supergroup	18 B	1	128 MB	t7_partition_1_0
这才是我们上传的文件						

当然我们也是可以手动创建一个分区的：

```
hive> alter table t7_partition_1 add partition(create_time='2015-12-25');
OK
Time taken: 0.556 seconds
```

如何查看我的表中有哪些分区呢，语法为： `show partitions tblName`

```
hive> show partitions t7_partition_1;
OK
create_time=2015-12-25
create_time=2015-12-26
Time taken: 0.425 seconds, Fetched: 2 row(s)
hive>
```

那同学的问题就来了，能增加一个分区，我能删除一个分区吗？必须是可以的，语法：

```
alter table tblName drop partition(partition_spec)
```

```
hive> alter table t7_partition_1 drop partition(create_time='2015-12-25');
Dropped the partition create_time=2015-12-25
OK
Time taken: 2.646 seconds
hive> show partitions t7_partition_1;
OK
create_time=2015-12-26
Time taken: 0.418 seconds, Fetched: 1 row(s)
hive>
```

实际上呢，删除分区实际上，是一并删除分区中的数据的。

在我们的 metastore 中呢？

PART_ID	CREATE_TIME	LAST_ACCESS_TIME	PART_NAME	SD_ID	TBL_ID
1	1451099235	0	create_time=2015-12-26	43	42

这些都是与 SDS、TBLS 互相对应的

有的同学还有问题，你能创建一个分区，有的业务有需求，我想创建多个分区呢？ of course，必须可以！

来，这里再举一个例子。某某学校，有若干学院，每年都招无数学生，唉，咱们的例子就可以这么来举，用年份和学院名称做分区，来，大家一起建。

```
create table t7_partition_2 (
    id int comment 'ID',
    name string comment 'name'
) partitioned by (year int comment 'admission year',
    school string comment 'school name')
row format delimited
fields terminated by '\t';

hive> create table t7_partition_2 (
    > id int comment 'ID',
    > name string comment 'name'
    > ) partitioned by (year int comment 'admission year', school string comment 'school name')
    > row format delimited
    > fields terminated by '\t';
OK
Time taken: 0.24 seconds
hive> desc t7_partition_2;
OK
id          int          ID
name        string        name
year        int          admission year
school      string        school name

# Partition Information
# col_name      data_type      comment
year          int          admission year
school        string        school name
Time taken: 0.434 seconds, Fetched: 10 row(s)
```

加载数据：

```
hive> load data local inpath '/home/crxxy/data/t7_partition_1_0' into table t7_partition_2 partition (year=2014,school='crxxy');
Loading data to table default.t7_partition_2 partition (year=2014, school=crxxy)
Partition default.t7_partition_2{year=2014, school=crxxy} stats: [numFiles=1, numRows=0, totalSize=18, rawDataSize=0]
OK
Time taken: 1.708 seconds
.....
```

查看分区：

```

hive> show partitions t7_partition_2;
OK
year=2014/school=arts
year=2014/school=crxy
year=2014/school=math
year=2015/school=arts
year=2015/school=crxy
year=2015/school=math
Time taken: 0.263 seconds, Fetched: 6 row(s)

```

查看 hdfs 目录：

Browse Directory

/user/hive/warehouse/t7_partition_2							Go!
Permission	Owner	Group	Size	Replication	Block Size	Name	
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	year=2014	
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	year=2015	
/user/hive/warehouse/t7_partition_2/year=2014							Go!
Permission	Owner	Group	Size	Replication	Block Size	Name	
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	school=arts	
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	school=crxy	
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	school=math	
/user/hive/warehouse/t7_partition_2/year=2014/school=arts							Go!
Permission	Owner	Group	Size	Replication	Block Size	Name	
-rw-r--r--	crxy	supergroup	18 B	1	128 MB	t7_partition_1_0	

说了如何创建、修改、增加、删除分区，重要的是我们改如何查询分区中的数据呢，唉，其实是非常的简单，分区呢相当于我们的一个查询条件，跟在 where 后面就可以了。

```

hive> select * from t7_partition_2;
OK
1      zhangsan    2014    arts      全表扫描, 这是时候分区就不起
2      lisi        2014    arts      作用了, 分区对表变成了透明的,
1      zhangsan    2014    crxy      或者说表忽略了分区
2      lisi        2014    crxy
1      zhangsan    2014    math
2      lisi        2014    math
1      zhangsan    2015    arts
2      lisi        2015    arts
1      zhangsan    2015    crxy
2      lisi        2015    crxy
1      zhangsan    2015    math
2      lisi        2015    math
Time taken: 2.129 seconds, Fetched: 12 row(s)
hive> select * from t7_partition_2 where school='arts';
OK
1      zhangsan    2014    arts      按照一个分区查询
2      lisi        2014    arts
1      zhangsan    2015    arts
2      lisi        2015    arts
Time taken: 0.948 seconds, Fetched: 4 row(s)
hive> select * from t7_partition_2 where school='arts' and year=2014;
OK
1      zhangsan    2014    arts      按照多个分区查询
2      lisi        2014    arts
Time taken: 0.493 seconds, Fetched: 2 row(s)

```

3.2.1.3.2. 关联 HDFS 中的数据到分区中

什么意思呢, 实际上就是如何引用 hdfs 中的数据到一张分区表。和前面学习过的外部表有点类似, 我们需要设置分区表引用的 hdfs 路径, 只不过写法不一样。

eg.

```

hive> alter table t7_partition_2 add partition(year=2016, school='chinese');
OK
Time taken: 0.702 seconds
hive> alter table t7_partition_2 partition(year=2016,school='chinese') set location "hdfs://master.hive.crxy.com:9000/data";
OK
Time taken: 1.006 seconds
hive> select * from t7_partition_2 where school='chinese';      在set之前呢, 必须得现有有分区存在
OK
1      zhangsan    2016    chinese
2      李四        2016    chinese
3      王五        2016    chinese
Time taken: 0.468 seconds, Fetched: 3 row(s)
hive> show partitions t7_partition_2;
OK
year=2014/school=arts
year=2014/school=crxy
year=2014/school=math
year=2015/school=arts
year=2015/school=crxy
year=2015/school=math
year=2016/school=chinese
Time taken: 0.349 seconds, Fetched: 7 row(s)

```

/user/hive/warehouse/t7_partition_2/year=2016/school=chinese

Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
------------	-------	-------	------	-------------	------------	------

hdfs hive 表中没有数据，我们删除

注意：

1) 在引用 hdfs 上面的数据是，必须得现有分区存在，不然就报错：

```
FAILED: SemanticException [Error 10006]: Partition not found  
{year=2017, school=chinese}
```

```
hive> alter table t7_partition_2 partition(year=2017,school='chinese') set location "hdfs://master.hive.crxy.com:9000/data";  
FAILED: SemanticException [Error 10006]: Partition not found (year=2017, school=chinese)
```

2) set location 的路径必须是 hdfs 的绝对路径，不然也会报错

```
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. {0} is not absolute or has no scheme information. Please specify a complete absolute uri with scheme information. /data
```

3.2.1.4. 桶表

我们知道，如果是分区表的话，可能数据会分的集中在某几个分区，其他分区数据不会很多，就像中国的人口，就主要集中河南、江苏、山东、广东、四川，其他省份就少的多了，你像西藏就三四百万，海南也挺少的，香港澳门就很少对不对，那这样对数据存储也不是很好，我们应该相对均匀的存放数据，每张表查询起来效率都差不多，是不是，这个时候我们就可以采用分桶。

桶表是对数据进行哈希取值，然后放到不同文件中存储。

分桶是将数据及分解成更容易管理的若干部分的另一种技术。

如果进行表连接操作，那么就需要对两张表的数据进行全扫描。非常耗费时间。可以针对连接字段进行优化。

这种情况下呢，对于相似的表中的数据进行比较的话就非常的方便了，只要对比相应的桶中的数据就可了。

如何建立一个桶表：

```
create table t8_bucket(id int) clustered by (id) into 3 buckets;
```

```

hive> create table t8_bucket (
    > id int comment 'ID'
    > ) clustered by(id) into 3 buckets;
OK
Time taken: 0.3 seconds
hive> desc t8_bucket; 按照哪一列来分桶，分成几个桶
OK
id                      int                         ID
Time taken: 0.377 seconds, Fetched: 1 row(s)

```

给桶中加载数据的时候，就不能使用 `load data` 的方式了，而是采用其它表中的数据，那么给他插入数据的写法就有新的变化了。

```
insert into table t8_bucket select * from t1 where id is not null;
```

注意，在插入数据之前需要先设置开启桶操作，不然插入数据不会设置为桶

```

hive> set hive.enforce.bucketing=true;

hive> insert into table t8_bucket select * from t1_bak where t_id is not null;
Query ID = crxy_20151226002626_4418dbaa-900c-4a5b-aabb-f17e550553e7
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 3
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:          走了mapreducer
  set mapreduce.job.reduces=<number>
Starting Job = job_1451031961293_0002, Tracking URL = http://master.hive.crxy.com:8080/jobs/job_1451031961293_0002
Kill Command = /home/crxy/deployed-soft/hadoop-2.4.1/bin/hadoop job -kill job_1451031961293_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 3
2015-12-26 00:27:48,388 Stage-1 map = 0%,  reduce = 0%
2015-12-26 00:28:48,685 Stage-1 map = 0%,  reduce = 0%
2015-12-26 00:29:07,198 Stage-1 map = 67%,  reduce = 0%, Cumulative CPU 4.58 sec
2015-12-26 00:29:10,327 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 9.92 sec

> select * from t8_bucket;
OK
3
3
4
1
4
1
5
2
5
2
Time taken: 0.596 seconds, Fetched: 10 row(s)

按照我们设置的桶数量3，来进行取模之后区分了，在hdfs中也存在了3个相应的文件

```

/user/hive/warehouse/t8_bucket

Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	crxy	supergroup	4 B	1	128 MB	000000_0
-rw-r--r--	crxy	supergroup	8 B	1	128 MB	000001_0
-rw-r--r--	crxy	supergroup	8 B	1	128 MB	000002_0

查看每个桶文件中的内容，可以看出是通过对 buckets 取模确定的。

桶表的主要作用：

数据抽样

提高某些查询效率

注意：

需要特别注意的是：clustered by 和 sorted by 不会影响数据的导入，这意味着，用户必须自己负责数据如何如何导入，包括数据的分桶和排序。

'set hive.enforce.bucketing = true'可以自动控制上一轮 reduce 的数量从而适配 bucket 的个数，当然，用户也可以自主设置 mapred.reduce.tasks 去适配 bucket 个数，推荐使用'set hive.enforce.bucketing = true'。

3.2.2. 视图

Hive 中，也有视图的概念，那我们都知道视图实际上是一张虚拟的表，是对数据的逻辑表示，只是一种显示的方式，主要的作用呢：

- 1、视图能够简化用户的操作
- 2、视图使用户能以多钟角度看待同一数据
- 3、视图对重构数据库提供了一定程度的逻辑独立性
- 4、视图能够对机密数据提供安全保护
- 5、适当的利用视图可以更清晰的表达查询

那我们在 Hive 中如何来创建一个视图呢？

```
create view t9_view as select * from t2;
```

```

hive> create view t9_view as select * from t2;
OK
Time taken: 1.939 seconds
hive> show tables;
OK
t1
t2
t2_2
t3_stu_hobby
t4
t4_stu_scores
t5_staff_address
t7_partition_1
t7_partition_2
t8_bucket
t9_view
Time taken: 0.183 seconds, Fetched: 11 row(s)

```

```

hive> desc t9_view;
OK
id                      int
stu_name                string
stu_birthday             date
online                  boolean
Time taken: 0.575 seconds, Fetched: 4 row(s)
hive> select * from t9_view;
OK
1      zhangsan      2014-01-13      true
2      李四        2015-11-11      false
3      王五        NULL        NULL
Time taken: 0.352 seconds, Fetched: 3 row(s)

```

/user/hive/warehouse

Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t1
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t2
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t2_2
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t3_stu_hobby
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t4
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t4_stu_scores
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t5_staff_address
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t7_partition_1
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t7_partition_2
drwxr-xr-x	crxy	supergroup	0 B	0	0 B	t8_bucket

元数据 metastore 中的体现

T..C..	L..O..R..	TBL NAME	TBL_TYPE	VIEW EXPANDED TEXT	VIEWS	VIEW ORIGIN
52 826	1 0 crx	0 57 t9_view	VIRTUAL_VIEW	select 't2_2'.`id`, 't2_2'.`stu_name`, 't2_2'.`stu_birthday` from 't2_2' where 't2_2'.`id` = 1;	99B	select * from t9_view
46 531	1 0 crx	0 55 t8_bucket	MANAGED_TABLE	(NULL)		OK (NULL)

3.2.3. 索引

创建索引

```
create index t1_index on table t1(stu_name) as
    'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
with deferred rebuild in table t1_index_table;

hive> create index t1_index on table t1(stu_name)
      > as 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler'
      > with deferred rebuild in table t1_index_table;
OK
Time taken: 0.713 seconds
```

as 指定索引器

重建索引

```
alter index t1_index on t1 rebuild;
```

显示索引

```
show formatted index on t1;
```

删除索引

```
drop index if exists t1_index on t1;
```

3.2.4. 数据加载语句

3.2.4.1. 从文件中装载

前面我们学习过如何从 linux fs 中加载数据到我们的 hive 表中，不知道大家注意到了没有，我们在写加载数据的语句的时候，一直要加一个关键词 local，如果我们不加呢，会怎么样，我们来看一下。

```
hive> load data inpath '/data/t2_4' into table t2;
FAILED: SemanticException Line 1:17 Invalid path ''/data/t2_4'': No files matching path hdfs://
master.hive.crxy.com:9000/data/t2_4' 我们没有加local的时候，它去hdfs目录里面寻找了!
hive>
```

大家来看一下，报错了。报错的信息显示，hdfs 中没有相应的数据文件，对不对，那么也就是说，如果我们不加入 local 关键字的话，它回到 hdfs 里面去查找数据文件，ok！

那我们就不加 local，从 hdfs 中传一个文件看一下。

没有 local：

```
hive> select * from t2;
OK
NULL      NULL      NULL      NULL
NULL      NULL      NULL      NULL
NULL      NULL      NULL      NULL
1        zhangsan    2014-01-13      true
NULL      NULL      NULL      NULL
NULL      NULL      NULL      NULL
Time taken: 1.852 seconds, Fetched: 6 row(s)

hive> dfs -ls /data;
Found 2 items
-rw-r--r--  1 crxy supergroup          75 2015-12-25 07:26 /data/t2
-rw-r--r--  1 crxy supergroup          75 2015-12-27 00:29 /data/t2_3
hive> dfs -text /data/t2_3;
1        zhangsan    2014-01-13      true
2        李四       2015-11-11      false
3        王五       2014-23-23      0
.

hive> load data inpath '/data/t2_3' into table t2;
Loading data to table default.t2
Table default.t2 stats: [numFiles=3, totalSize=225]
OK
Time taken: 1.461 seconds 现在目录下面有3个文件了
hive>
```

/user/hive/warehouse/t2

Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	crxy	supergroup	75 B	1	128 MB	t2
-rw-r--r--	crxy	supergroup	75 B	1	128 MB	t2_2
-rw-r--r--	crxy	supergroup	75 B	1	128 MB	t2_3

那有个问题，hdfs 上面的数据还在吗？

```
hive> dfs -ls /data;
Found 1 items
-rw-r--r--  1 crxy supergroup          75 2015-12-25 07:26 /data/t2
hive>
```

我们发现数据不在了，相当于**数据的移动** (hdfs dfs -mv uri1 uri2)

我们再来看一下，其实我们都已经上传的这几份文件的内容都是一样的，我们不想重复上传，只要一份，这样改怎么办呢？其实也非常简单，使用关键字 `overwrite`，我们来看，从 linux 本地上传一份数据，为了上传后有数据显示，我们使用下面的数据作为数据源：

```
[crxy@master data]$ more t2_2;
1 zhangsan 2014-01-13 true
2 李四 2015-11-11 false
3 王五 2014-23-23 0
```

```
load data local inpath '/home/crxy/data/t2_2' overwrite into table t2;
```

```
hive> load data local inpath '/home/crxy/data/t2_2' overwrite into table t2;
Loading data to table default.t2
Table default.t2 stats: [numFiles=1, numRows=0, totalSize=75, rawDataSize=0]
OK
Time taken: 0.974 seconds
hive> t2表中的文件变成一个了！！！
```

表中数据

```
hive> select * from t2;
OK
1 zhangsan 2014-01-13 true
NULL NULL NULL NULL
NULL NULL NULL NULL
Time taken: 0.281 seconds, Fetched: 3 row(s)
hive> [green]
```

/user/hive/warehouse/t2

Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	crxy	supergroup	75 B	只留下一份数据了	128 MB	t2_2

ok, 实际上 **overwrite** 干了一件什么事, 将原来的数据删除后, 在将新的数据上传。

还有我们前面学习过分区表, 如何将数据加载到分区表中的某一个分区中, 这里就不多说, 我们来简单的总结一下, 从文件加载数据的语法。

```
load data [local] inpath ... [overwrite] into table ... [partition
(....)]
```

[local] 指的是 linux 目录文件, 如果没有 **local**, 则指的是 hdfs。如果路径是 hdfs 的话, 那么文件被移动到 hive 目录下。

[overwrite] 使用之后就可以覆盖原始数据文件, 只保留一份。如果不使用, 则追加到原来的表中, 相当于在 hdfs 中增加一个数据文件。

[partition (....)] 表示分区

3.2.4.2. 从其他表中装载

大家都学习过 mysql 的相关语法，同 mysql 相似的，在 hive 中也可从其他表中来装载数据，非常的方便。

```
insert [into/overwrite] table t1 select columns... from t2;
```

```

hive> insert into table t1 select id from t2;
Query ID = crxy_20151227010909_2fccefca4-fc77-421e-932b-a54ed6ae6f6f
Total jobs = 3
Launching Job 1 out of 3          我们的操作转换成了mapreduce
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1451190835063_0001, Tracking URL = http://master.hive.crxy.com:8088/proxy/application_1451190835063_0001/
Kill Command = /home/crxy/deployed-soft/hadoop-2.4.1/bin/hadoop job -kill job_1451190835063_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2015-12-27 01:10:23,125 Stage-1 map = 0%, reduce = 0%
2015-12-27 01:10:58,449 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 5.28 sec
MapReduce Total cumulative CPU time: 5 seconds 280 msec
Ended Job = job_1451190835063_0001
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: hdfs://master.hive.crxy.com:9000/tmp/hive/crxy/3752bdf3-4464-4074-81b4-6498b664e375/hive_2015-12-27_01/-ext-10000
Loading data to table default.t1          告诉我们数据发生了变化
Table default.t1 stats: [numFiles=4, numRows=3, totalSize=103, rawDataSize=5]
MapReduce Jobs Launched: [numFiles=4]      在原来的基础上多了一个文件
Stage-Stage-1: Map: 1  Cumulative CPU: 5.28 sec  HDFS Read: 289 HDFS Write: 74 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 280 msec
OK
Time taken: 92.35 seconds

hive> select * from t1;
OK
1
NULL
NULL
1
2
3
4
5
1
2
3
4
5
NULL
NULL
NULL
Time taken: 0.177 seconds, Fetched: 16 row(s)

```

我们看到，数据被加载进去了，hdfs 中也多了一份拷贝的数据文件。

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	crxy	supergroup	8 B	1	128 MB	000000_0
-rw-r--r--	crxy	supergroup	10 B	1	128 MB	t1
-rw-r--r--	crxy	supergroup	10 B	1	128 MB	t1_1
-rw-r--r--	crxy	supergroup	75 B	1	128 MB	t2

这里说明一点，被装在的表的列必须要和 select 后的选择的列一致。这里我们也可以使用 overwrite 来覆盖掉原来的数据，eg.

```
hive> insert overwrite table t1 select id from t2;
Query ID = crxy_20151227011616_9b949b7c-dbec-472c-9f53-54ec58ca20bf
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
转成了mapreducer来执行任务
```

t1 被覆盖了。

```
hive> select * from t1;
OK
1
NULL
NULL
```

3. 2. 4. 3. 通过动态分区装载数据

不开启只支持

```
hive>INSERT OVERWRITE TABLE t3 PARTITION(province='bj', city)
SELECT t.province, t.city FROM temp t WHERE t.province='bj';
```

开启动态分区支持

```
hive>set hive.exec.dynamic.partition=true;
hive>set hive.exec.dynamic.partition.mode=strict;
hive>set hive.exec.max.dynamic.partitions.perNode=1000;
#加载语句
hive>INSERT OVERWRITE TABLE t3 PARTITION(province, city)
SELECT t.province, t.city FROM temp t;
```

3. 2. 4. 4. 建立表的同时装载数据

前面我们同样学习使用过创建表的时候装载数据的案例，比如视图，下面就给大家再来简单的说一下说一下创建表的时候装载数据的语法。

```
CREATE TABLE new_tblName AS SELECT [CLOUMNS...] FROM other_tblName.
```

3.2.5. 数据导出

1°、在 hdfs 复制文件(夹)

```
hadoop fs -cp src_uri dest_uri
```

2°、使用 DIRECTORY

```
hive>INSERT OVERWRITE [LOCAL] DIRECTORY '...' SELECT ...FROM...WHERE ...;
```

```
hive> insert overwrite local directory 'data/t1_out' select * from t1;
Automatically selecting local only mode for query
Query ID = crxy_20151231003737_7fd74066-2589-4ae3-a78f-8951553717d9
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
```

```
hive> !ls -l data/t1_out;
total 4
-rw-r--r--. 1 crxy crxy 30 Dec 31 00:37 000000_0.gz
```

3.2.6. 本地模式

唉我们，来看，有的同学感觉每次都执行 MR 的速度非常的慢，心情非常不爽，那有没有办法让 hql 的执行速度加快一点呢？必须有，怎么弄？开启 hql 执行模式为本地模式，这样就哦了！

可以 **set hive.exec.mode.local.auto=true**; 来开启(默认为 false) 我们再来执行一下前面的 hql 语句

```
hive> insert overwrite table t1 select id from t2;
Automatically selecting local only mode for query
Query ID = crxy_20151227012929_c1cdf819-83f5-4c17-b233-ccdf0b55514a
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Job running in-process (local Hadoop)
Hadoop job information for Stage-1: number of mappers: 0; number of reducers
2015-12-27 01:30:03,238 Stage-1 map = 100%, reduce = 0%
Ended Job = job_local714986_0001
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: hdfs://master.hive.crxy.com:9000/tmp/hive/crxy/3752bdf3-4464-1/-ext-10000
Loading data to table default.t1
Table default.t1 stats: [numFiles=1, numRows=3, totalSize=8, rawDataSize=5]
MapReduce Jobs Launched:
Stage-Stage-1: HDFS Read: 1174 HDFS Write: 35135036 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 8.917 seconds
hive>
```

耗时才 9 秒不到，足足比前面快了近 8 倍。

可以使用 **set hive.exec.mode.local.auto** 查看是否开启了本地模式。

如果我们的测试数据集非常小，那么没必要产生 MR job，使用本地模式吧。在集群生产的时候不建议使用本地模式，这会使其中一台机器作业量剧增，会累垮某一台机器的。

3. 3. 考题

问：什么时候使用外部表，什么时候用内部表？

一般公用的一些数据使用外部表，比如多个部门都用的一些统计数据，用外之后就可以删除掉表的定义，而不用影响表的数据。从上面也可以看出来，我们使用别人的数据时候，最好来建立这种外部表。

问：如何建立一个外部分区表？

问：如何开启本地模式？

3. 4. 实验

第 4 次课 Hive 高阶查询

4. 1. 教学目标

4. 1. 1. Hive 的查询条件

4. 1. 2. Hive 中的 MapReduce 任务

4. 1. 3. 控制 Hive 中 MR 数量

4. 1. 4. Hive 中排序

4. 2. 课程内容

4. 2. 1. 条件显示

我们知道，hive 主要是来做数理统计分析的，所以经常会需要对我们的计算结果进行各种汇总分析，处理，比如处理为 null 的结果，比如对获取的 id 进行分类显示等等，

我们知道，在 mysql 中使用 ifnull(field, replaceVal) 来过滤结果，输出我们满意的结果，eg.

The screenshot shows the MySQL Workbench interface. At the top, a query is written:

```
1     SELECT id, IFNULL(NAME, 'lucy') NAME, age, gender FROM person;
```

The result set is displayed in two tabs: "Table Data" and "Result".

	id	age	name	gender
1	1	18	jackcli	male
2	2	12	(NULL)	female

	id	name	age	gender
1	1	jackcli	18	male
2	2	lucy	12	female

那么，我们在 hive 中应该处理这类的问题呢，这里再复杂一点，对多个结果进行分类显示，该怎么办呢，哎，其实也是非常的简单，我们使用

`case field (when val then res) else res_other end` 这个语法，来，这里举一个例子：

```
set hive.cli.print.header=true;
```

使用 `set hive.cli.print.header=true;` 显示列名称

```
hive> select * from t1;
OK
t1.id
1
2
3
4
5
Time taken: 0.261 seconds, Fetched: 5 row(s)
hive> select id, case id
> when 1 then 'staff'
> when 2 then 'student'
> when 3 then 'teacher'
> when 4 then 'boss'
> else 'other' end
> from t1;
OK
id      _c1
1      staff
2      student
3      teacher
4      boss
5      other
Time taken: 0.526 seconds, Fetched: 5 row(s)
```

使用 `case when then` 对结果进行加工

4.2.2. Hive 中的 MapReduce 任务

4.2.2.1. 什么情况下 Hive 可以避免 MR

来说一下，我们的 HQL 语句有时会转化为 MR 执行，有时候不会转化为 MR 执行，我们这里来简单的说一下，什么的情况下会转换为 MR，什么情况不容易转化为 MR

在进行全表扫描的时候，可以用 hadoop fs -text/hadoop fs -tail 命令代替的方式，是不会走 mapreduce，就是咱们的 select * from t，这个可以分为两种情况。

第一种情况：普通的表

```
select * from tblName [limit limit_num];
```

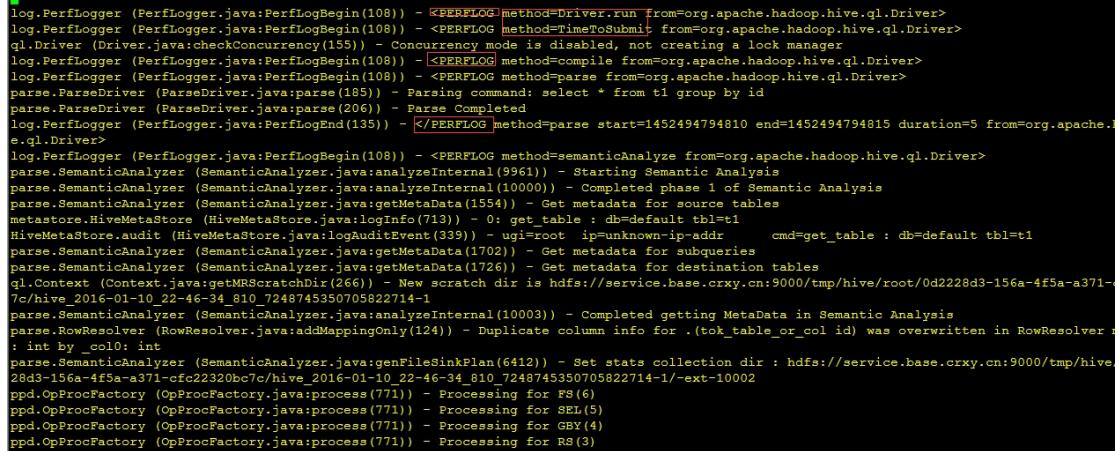
第二种情况：分区

```
select * from tblName where partition_sec [limit limit_num];
```

其余基本都会走 MR 的。

4.2.2.2. MR 的执行过程

我们可以通过查看 hql 执行过程中的日志来粗略的了解一下 hive 和 hadoop 的关系：



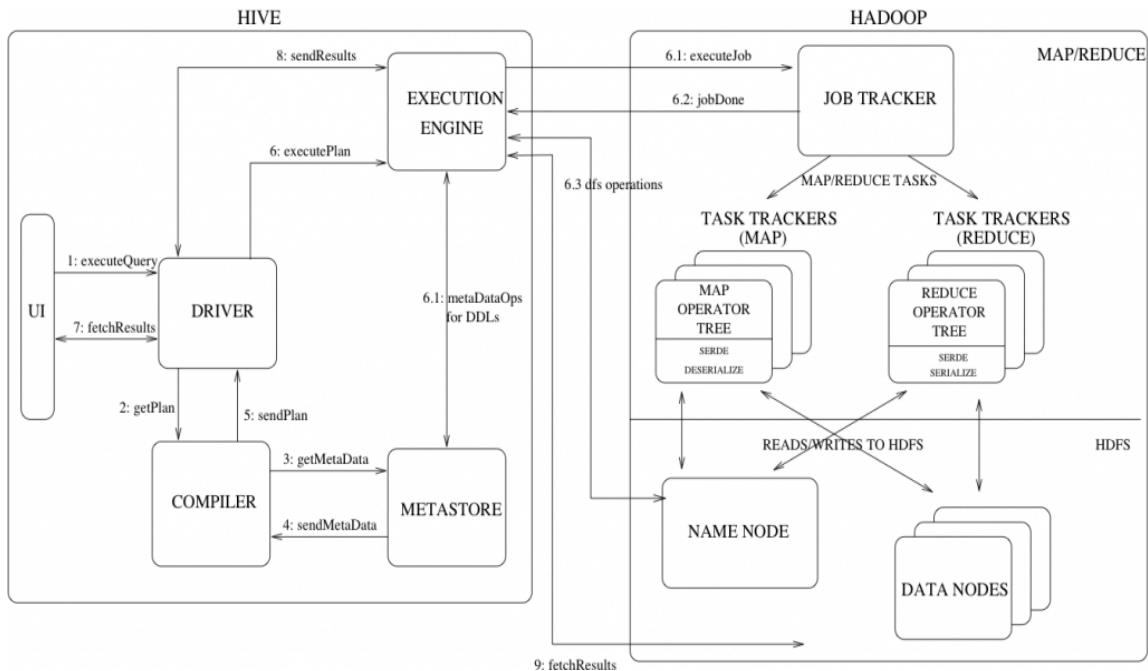
```
log.PerfLogger (PerfLogger.java:PerfLogBegin(108)) - <PERFLOG method=Driver.run from=org.apache.hadoop.hive.ql.Driver>
log.PerfLogger (PerfLogger.java:PerfLogBegin(108)) - <PERFLOG method=TimeToSubmit from=org.apache.hadoop.hive.ql.Driver>
ql.Driver (Driver.java:checkConcurrency(155)) - Concurrency mode is disabled, not creating a lock manager
log.PerfLogger (PerfLogger.java:PerfLogBegin(108)) - <PERFLOG method=compile from=org.apache.hadoop.hive.ql.Driver>
log.PerfLogger (PerfLogger.java:PerfLogBegin(108)) - <PERFLOG method=parse from=org.apache.hadoop.hive.ql.Driver>
parse.ParseDriver (ParseDriver.java:parse(185)) - Parsing command: select * from t1 group by id
parse.ParseDriver (ParseDriver.java:parse(206)) - Parse Completed
log.PerfLogger (PerfLogger.java:PerfLogEnd(135)) - </PERFLOG method=parse start=1452494794810 end=1452494794815 duration=5 from=org.apache.hadoop.hive.ql.Driver>
log.PerfLogger (PerfLogger.java:PerfLogBegin(108)) - <PERFLOG method=semanticAnalyze from=org.apache.hadoop.hive.ql.Driver>
parse.SemanticAnalyzer (SemanticAnalyzer.java:analyzeInternal(9961)) - Starting Semantic Analysis
parse.SemanticAnalyzer (SemanticAnalyzer.java:analyzeInternal(10000)) - Completed phase 1 of Semantic Analysis
parse.SemanticAnalyzer (SemanticAnalyzer.java:getMetaData(1554)) - Get metadata for source tables
metastore.HiveMetaStore (HiveMetaStore.java:logInfo(713)) - 0: get_table : db=default tbl=t1
HiveMetaStore.audit (HiveMetaStore.java:logAuditEvent(399)) - ugi=root ip=unknown-ip-addr cmd=get_table : db=default tbl=t1
parse.SemanticAnalyzer (SemanticAnalyzer.java:getMetaData(1702)) - Get metadata for subqueries
parse.SemanticAnalyzer (SemanticAnalyzer.java:getMetaData(1726)) - Get metadata for destination tables
ql.Context (Context.java:getMRScratchDir(266)) - New scratch dir is hdfs://service.base.crxy.cn:9000/tmp/hive/0d2228d3-156a-4f5a-a371-7c/hive_2016-01-10_22-46-34_810_7248745350705822714-1
parse.SemanticAnalyzer (SemanticAnalyzer.java:analyzeInternal(10003)) - Completed getting MetaData in Semantic Analysis
parse.RowResolver (RowResolver.java:addMappingOnly(124)) - Duplicate column info for .(tbl_table_or_col_id) was overwritten in RowResolver
: int by _col0: int
parse.SemanticAnalyzer (SemanticAnalyzer.java:genFileSinkPlan(6412)) - Set stats collection dir : hdfs://service.base.crxy.cn:9000/tmp/hive/28d3-156a-4f5a-a371-cfc2320bc7c/hive_2016-01-10_22-46-34_810_7248745350705822714-1/-ext-10002
ppd.OpProcFactory (OpProcFactory.java:process(771)) - Processing for FS(6)
ppd.OpProcFactory (OpProcFactory.java:process(771)) - Processing for SEL(5)
ppd.OpProcFactory (OpProcFactory.java:process(771)) - Processing for GBY(4)
ppd.OpProcFactory (OpProcFactory.java:process(771)) - Processing for RS(3)
```



截图中不断出现一个可以从 method 以及不断出现的类 org.apache.hadoop.hive.ql.Driver 来看出我们 Hive 驱动的执行情况，了解上述的东西之后呢，可以简单的通过源码来看一下。

Driver 执行作业调度了 org.apache.hadoop.hive.ql.exec.Task

看过了上述的东西之后呢，咱们再来看之前提到过的 Hive 和 Hadoop 的调用关系，应该就相对简单的多了。



4.2.3. 控制 Hive 中 MR 数量

通常情况下，作业会通过 `input` 的目录产生一个或者多个 map 任务。

4.2.3.1. 控制 Map Task 的数量

Mapper task 数量有 `inputsplit` 数量决定。Block size 由参数 `dfs.block.size` 决定。

但是执行 mapper task 时使用的 `InputFormat` 是由 `hive.input.format` 参数决定的。参数的值是 `org.apache.hadoop.hive.ql.io.CombineHiveInputFormat`，意味着在执行 mapper task 时，会对大量的小文件进行合并。

问：如果有 2 个 20MB 的小文件，那么产生几个 mapper task？如果有 700MB 的文件，产生几个？

作业：有 3 个文件，分别是 10m、20、130m，产生几个？假设 `set dfs.block.size=64MB`；再看一下运行结果。

4.2.3.2. 控制 Reduce Task 的数量

参数 `mapred.reduce.tasks`（默认为 -1）如果是负数，那么推测 reducer 任务数

量。

参数 1 `hive.exec.reducers.bytes.per.reducer` 决定每个 reducer task 可以处理的最大数据量， 默认是 256MB。

参数 2 `hive.exec.reducers.max` 决定了最大可以只写的任务数量， 默认是 1009.

max number of reducers will be used. If the one specified in the configuration parameter `mapred.reduce.tasks` is negative, Hive will use this one as the max number of reducers when automatically determine number of reducers. (当 `mapred.reduce.tasks` 值为负数时，会自动设置 reduce 的数量，但是是有上限的最大为 `hive.exec.reducers.max`)

计算 reducer 数的公式很简单 $N = \min(\text{参数 2}, \text{总输入数据量} / \text{参数 1})$

依据 Hadoop 的经验，可以将参数 2 设定为 $0.95 * (\text{集群中 TaskTracker 个数})$ 。

正确的 reduce 任务的个数应该是 0.95 或者 $1.75 \times (\text{节点数} \times \text{mapred.tasktracker.tasks.maximum} \text{ 参数值})$ 。如果任务数是节点个数的 0.95 倍，那么所有的 reduce 任务能够在 map 任务的输出传输结束后同时开始运行。如果任务数是节点个数的 1.75 倍，那么高速的节点会在完成他们第一批 reduce 任务计算之后开始计算第二批 reduce 任务，这样的情况更有利于负载均衡。

4.2.4. `reduce` 数量只有一个

当我们在写 hql 语句的时候，一般会产生 MR 程序，但是它产生的 reducer task 数量却只有一个，这个比较恶心，map 结束之后的数据全都汇聚到一个节点上来，这样就和分布式计算相悖了，使得一个节点承受的压力过大，使我们的 hql 语句执行效率变低，对我们的程序而言是不好的结果，我们应该尽量让其在多个节点上执行。

那怎么才能让它在多个节点上执行呢，设置 `mapred.reduce.tasks`，当然这个不一定起作用，因为可能，再具体的环境下业务优先。

什么情况下会有 reduce 数量只有一个呢？

1° 进行汇总操作的时候，没有使用 `group by` 分组

2° 使用了子句 `order by` 排序（全局排序，只能在单节点进行排序）

3° 笛卡尔积 $M * N$ （表连接，但是没有 `on` 条件）

查看属性: `hive.exec.parallel`(是否并行执行 mr jobs)

```
hive> set hive.exec.parallel;
hive.exec.parallel=false
hive> set hive.exec.parallel.thread.number;
hive.exec.parallel.thread.number=8
hive>
```

这个东西没有统一的定论，和我们每一个具体环境，数据、集群环境、内存、cpu、磁盘等等都有关系，不是一概而论的，需要我们在今后的工作中，不断的调试找出最优的一套配置方案，这实际上就是我们 hive 的调优。

4.2.5. Hive 中排序

排序，这个概念在我们平时的使用过程中非常的广泛，在数理统计的领域里面尤甚，我们知道在 mysql 中使用的 `order by` 字段，在 hive 中呢，也是一样的，当然还有 hive 中的一些小的改动。

`order by` 排序使用全局排序，只有一个 reducer task 运行。

```
hive> select * from t4;
OK
t4.id      t4.name      t4.scores
1          zhangsan      {"chinese":60,"math":61,"english":62}
2          lisi          {"chinese:90":null,"math:60":null,"english:32":null}
.

hive> select * from t4 order by id desc;
Query ID = crxy_20151229052222_7c421751-7119-4c17-abc7-7ab6a1f280f6
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_1451357225044_0001, Tracking URL = http://master.hive.crxy.com:8088/proxy/application_1451357225044_0001/
Kill Command = /home/crxy/deployed-soft/hadoop-2.4.1/bin/hadoop job -kill job_1451357225044_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2015-12-29 05:23:47,515 Stage-1 map = 0%, reduce = 0%
2015-12-29 05:24:34,219 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 17.28 sec
2015-12-29 05:24:44,090 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 19.83 sec
MapReduce Total cumulative CPU time: 19 seconds 830 msec
Ended Job = job_1451357225044_0001
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 19.83 sec    HDFS Read: 290 HDFS Write: 87 SUCCESS
Total MapReduce CPU Time Spent: 19 seconds 830 msec
OK
t4.id      t4.name      t4.scores
2          lisi          {"chinese:90":null,"math:60":null,"english:32":null}
1          zhangsan      {"chinese":60,"math":61,"english":62}
Time taken: 116.459 seconds, Fetched: 2 row(s)
```

`sort by` 排序是在每个 reducer task 里面进行局部排序，不做全局排序。

```
hive> select * from t7_partition_2 sort by id desc;
Query ID = crxy_20151229052929_52035eb0-5ca0-4d38-b5ab-6c301f4221ab
```

```
t7_partition_2.id      t7_partition_2.name      t7_partition_2.year      t7_partition_2.school
3      王五      2016      chinesee
2      李四      2016      chinesee
2      lisi       2015      math
2      lisi       2015      crxy
2      lisi       2015      arts
2      lisi       2014      math
2      lisi       2014      crxy
1      zhangsan   2015      math
1      zhangsan   2015      crxy
1      zhangsan   2015      arts
1      zhangsan   2014      math
1      zhangsan   2014      crxy
1      zhangsan   2016      chinesee
```

distribute by ... sort by ...

cluster by ...是distribute by ... sort by ...的简写。

4. 3. 考题

4. 4. 实验

第 5 次课 Hive 高阶应用—函数

5.1. 教学目标

5.1.1. Hive 中数据的存储类型

5.1.2. Hive 中的内嵌函数

5.1.3. Hive 自定义函数

5.1.4. Hive JDBC

5.2. 课程内容

5.2.1. Hive 中文件的存储类型

hive 中有哪些文件的存储类型呢, 我们可以通过 `hive-site.xml` 配置文件的配置项 (`hive.default.fileformat`) 来看一下:

```
<property>
  <name>hive.default.fileformat</name>
  <value>TextFile</value>
  <description>
    Expects one of [textfile, sequencefile, rcfile, orc].
    Default file format for CREATE TABLE statement. Users can explicitly override it by
    CREATE TABLE ... STORED AS [FORMAT]
  </description>
</property>
```

来我们下面就说说文件存储类型的事

```
hive> create database stored_type;
OK
Time taken: 0.353 seconds
hive> use stored_type;
OK
Time taken: 0.078 seconds
```

5.2.1.1. TextFile

Hive 默认格式, 数据不做压缩, 磁盘开销大, 数据解析开销大。可结合 Gzip、Bzip2、Snappy 等使用 (系统自动检查, 执行查询时自动解压), 但使用这种方式,

hive 不会对数据进行切分，从而无法对数据进行并行操作。

```

hive> create table t12_text(
    > id int comment 'ID',
    > name string comment 'name',
    > province string comment 'province',
    > )row format delimited
    > fields terminated by '\t'
    > stored as textfile;
OK
Time taken: 0.326 seconds

hive> set hive.exec.compress.output=true;
hive> set mapred.output.compress=true;
hive> set mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec;
hive> set io.compression.codec=org.apache.hadoop.io.compress.GzioCodec;
hive> load data local inpath 'data/t12_text' into table t12_text;
Loading data to table stored_type.t12_text
Table stored_type.t12_text stats: [numFiles=1, totalSize=73]
OK
Time taken: 1.544 seconds
hive> select * from t12_text;
OK
t12_text.id      t12_text.name      t12_text.province
1      jack      gansu
2      bella     hubei
3      superwu   beijing
4      leibusi   hubei
5      mao       hunan
Time taken: 0.289 seconds, Fetched: 5 row(s)

```

5.2.1.2. SequenceFile

SequenceFile 是 Hadoop API 提供的一种二进制文件支持，其具有使用方便、可分割、可压缩的特点。

SequenceFile 支持三种压缩选择：NONE，RECORD，BLOCK。Record 压缩率低，一般建议使用 BLOCK 压缩。

```

TIME taken: 1.783 seconds
hive> create table t12_sequence(
    > id int comment 'ID',
    > name string comment 'name',
    > province string comment 'province',
    > )row format delimited
    > fields terminated by '\t'
    > stored as sequencefile;
OK
Time taken: 0.293 seconds
hive> set mapred.output.compression.type=BLOCK;
hive> insert overwrite table t12_sequence select * from t12_text;
Automatically selecting local only mode for query
Query ID = crxy_20151231000000_394c5322-d5f3-4afb-adaf-563874733b58
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Job running in-process (local Hadoop)

```

注：如果使用 32bit 的 hadoop，会有问题!!! 点我

5.2.1.3. RCFile

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

```
hive> create table t12_rcfile(
  > id int comment 'ID',
  > name string comment 'name',
  > province string comment 'province'
  > ) row format delimited
  > fields terminated by '\t'
  > stored as rcfile;
OK
Time taken: 0.411 seconds
hive> insert overwrite table t12_rcfile select * from t12_text;
Automatically selecting local only mode for query
Query ID = crxy_20151231000606_2bf63e94-40ea-422b-a79b-271d3fb698f0
Total jobs = 1
Launching Job 1 out of 1
```

结果对比

```
hive> dfs -ls -R hdfs://master.hive.crxy.com:9000/user/hive/warehouse/stored_type.db;
Time taken: 77.151 seconds
drwxr-xr-x   0  crxy  supergroup  0 2015-12-31 00:23 hdfs://master.hive.crxy.com:9000/user/hive/warehouse/stored_type.db/t12_rcfile
-rw-r--r--   1  crxy  supergroup  316795 2015-12-31 00:23 hdfs://master.hive.crxy.com:9000/user/hive/warehouse/stored_type.db/t12_rcfile/000000_0
drwxr-xr-x   0  crxy  supergroup  0 2015-12-31 00:25 hdfs://master.hive.crxy.com:9000/user/hive/warehouse/stored_type.db/t12_sequence
-rw-r--r--   1  crxy  supergroup  612040 2015-12-31 00:25 hdfs://master.hive.crxy.com:9000/user/hive/warehouse/stored_type.db/t12_sequence/000000_0
drwxr-xr-x   0  crxy  supergroup  0 2015-12-31 00:22 hdfs://master.hive.crxy.com:9000/user/hive/warehouse/stored_type.db/t12_text
-rw-r--r--   1  crxy  supergroup  146206073 2015-12-31 00:22 hdfs://master.hive.crxy.com:9000/user/hive/warehouse/stored_type.db/t12_text/t12_text

hive> select * from t12_text limit 1000;
+-----+-----+
| id  | name |
+-----+-----+
| 3   | superwu beijing |
+-----+-----+
Time taken: 0.378 seconds, Fetched: 1000 row(s)

hive> select * from t12_sequence limit 1000;
+-----+-----+
| id  | name |
+-----+-----+
| 3   | superwu beijing |
+-----+-----+
Time taken: 0.333 seconds, Fetched: 1000 row(s)

hive> select * from t12_rcfile limit 1000;
+-----+-----+
| id  | name |
+-----+-----+
| 3   | superwu beijing |
+-----+-----+
Time taken: 0.326 seconds, Fetched: 1000 row(s)
```

5.2.1.4. 总结

textfile 存储空间消耗比较大，并且压缩的 text 无法分割和合并 查询的效率最低，可以直接存储，加载数据的速度最高

sequencefile 存储空间消耗大，压缩的文件可以分割和合并 查询效率高，需要通过 text 文件转化来加载

rcfile 存储空间最小，查询的效率最高，需要通过 text 文件转化来加载，加载的

速度最低

相比 TEXTFILE 和 SEQUENCEFILE, RCFILE 由于列式存储方式, 数据加载时性能消耗较大, 但是具有较好的压缩比和查询响应。数据仓库的特点是一次写入、多次读取, 因此, 整体来看, RCFILE 相比其余两种格式具有较明显的优势。

5.2.2. Hive 中的内嵌函数

同 mysql 一样的, 因为 hive 也是一个主要做统计的软件, 所以为了满足各种各样的统计需要, 其也内置了相当多的函数, 我们可以通过 show functions; 来查看 hive 的内置函数

```
hive> show functions;
OK
tab_name
!
!=
%
&
*
+
-
/
<
```

查看具体的函数我们可以使用相应的命令: desc function functionName;

eg:

```
hive> desc function year;
OK
tab_name      需要一个日期参数, 返回日期的年份
year(date) - Returns the year of date
Time taken: 0.072 seconds, Fetched: 1 row(s)
```

都会来详细的说明参数, 及其返回值。

when

```
hive> desc function extended when;
OK
tab_name
CASE WHEN a THEN b [WHEN c THEN d]* [ELSE e] END - When a = true, returns b; when c = true, return d; else return e
Example:
SELECT
CASE
    WHEN deptno=1 THEN Engineering
    WHEN deptno=2 THEN Finance
    ELSE admin
END,
CASE
    WHEN zone=7 THEN Americas
    ELSE Asia-Pac
END
FROM emp_details
Time taken: 0.106 seconds, Fetched: 13 row(s)
```

concat

```
hive> desc function extended concat;
OK
tab_name
concat(str1, str2, ... strN) - returns the concatenation of str1, str2, ... strN or concat(bin1, bin2, ... binN) - returns the concatenation of bytes in binary data bin1, bin2, ... binN
Returns NULL if any argument is NULL.
Example:
> SELECT concat('abc', 'def') FROM src LIMIT 1;
'abcdef'
Time taken: 0.081 seconds, Fetched: 5 row(s)
hive>
```

collect_list

```
hive> desc function extended collect_list;
OK
tab_name
collect_list(x) - Returns a list of objects with duplicates
Time taken: 0.18 seconds, Fetched: 1 row(s)
```

collect_set

```
hive> desc function extended collect_set;
OK
tab_name
collect_set(x) - Returns a set of objects with duplicate elements eliminated
Time taken: 0.093 seconds, Fetched: 1 row(s)
```

split

```
hive> desc function extended split;
OK
tab_name
split(str, regex) - Splits str around occurrences that match regex
Example:
> SELECT split('oneAtwoBthreeC', '[ABC]') FROM src LIMIT 1;
["one", "two", "three"]
Time taken: 0.094 seconds, Fetched: 4 row(s)
```

explode

```
hive> desc function extended explode;
OK
tab_name
explode(a) - separates the elements of array a into multiple rows, or the elements of a
map into multiple rows and columns
Time taken: 0.1 seconds, Fetched: 1 row(s)
```

举例：完成单词计数

数据准备：

```
hive> select * from t10_wc;
OK
t10_wc.word
hello bella
hello dachui
hello jack
Time taken: 0.264 seconds, Fetched: 3 row(s)
```

第一步，将数据切割成一个一个的单词，使用 split 函数

```
hive> select split(word, ' ') from t10_wc;
OK
_c0
["hello","bella"]
["hello","dachui"]
["hello","jack"]
Time taken: 0.359 seconds, Fetched: 3 row(s)
```

第二步，将集合中的元素拆分成多行元素，使用 explode 函数

```
select explode(split(word, ' ')) from t10_wc;
```

```
hive> select explode(split(word, ' ')) from t10_wc;
Query ID = crxy_20151230061313_80a9dddc-07c7-49a5-a258-229a5cafcdc95
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1451357225044_0013, Tracking URL = http://master.hive.crxy.com:8088/proxy/api/jobs/job_1451357225044_0013
Kill Command = /home/crxy/deployed-soft/hadoop-2.4.1/bin/hadoop job -kill job_1451357225044_0013
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2015-12-30 06:13:55,641 Stage-1 map = 0%, reduce = 0%
2015-12-30 06:14:04,359 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 3.07 sec
MapReduce Total cumulative CPU time: 3 seconds 70 msec
Ended Job = job_1451357225044_0013
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1 Cumulative CPU: 3.07 sec HDFS Read: 256 HDFS Write: 46 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 70 msec
OK
col
hello
bella
hello
dachui
hello
jack
Time taken: 34.216 seconds, Fetched: 6 row(s)
```

第三步，再使用聚合函数统计多行数据，so easy。

```
select wc.word, count(1) as count from
(select explode(split(word, ' ')) as word from t10_wc) wc
group by wc.word order by count desc;
```

```

hive> select wc.word, count(1) as count from
    > (select explode(split(word, ' ')) as word from t10_wc) wc
    > group by wc.word
    > order by count desc;
Automatically selecting local only mode for query
Query ID = crxy_20151230062727_670302fa-cf54-4707-8d79-7cfcff0de4cd
Total jobs = 2
Launching Job 1 out of 2
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>
...
wc.word count
hello      3
jack       1
dachui     1
bella      1

```

行转列

```

select u.id, u.name, a.address from t11_user u join t11_address a on u.id = a.uid;

Stage-Stage-3: HDFS Read: 778 HDFS Write: 669 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
u.id      u.name    a.address
2         lisi      henan
1         zhangsan   gansu
3         wangwu    beijing
1         zhangsan   shangdong
Time taken: 30.243 seconds, Fetched: 4 row(s)

```

第二步

```

hive> select max(u.id), u.name, concat_ws(",", collect_set(a.address)) from t11_user u join t11_address a on u.id = a.uid group by u.id;
FAILED: SemanticException [Error 10002]: Line 1:20 Invalid column reference 'name'

```

出现错误，因为有多个 name 不知以哪个为准，所以用聚合函数过滤一下

```

select u.id as id, max(u.name) as name, concat_ws(",", collect_set(a.address))

as addr from t11_user u join t11_address a on u.id = a.uid group by u.id;

hive> select u.id as id, max(u.name) as name, concat_ws(",", collect_set(a.address)) as addr
    > from t11_user u join t11_address a on u.id = a.uid
    > group by u.id;
Automatically selecting local only mode for query
Query ID = crxy_20151230230606_129a9ddc-2e85-4a24-a79b-074a5dff7e0
Total jobs = 1

Total MapReduce CPU Time Spent: 0 msec
OK
id      name      addr
1       zhangsan   shangdong,gansu
2       lisi       henan
3       wangwu    beijing
Time taken: 31.648 seconds, Fetched: 3 row(s)

```

列转行：

数据准备

```
Time taken: 25.714 seconds, Fetched: 3 row(s)
hive> create table t12_user_addr as
  > select u.id as id, max(u.name) as name, concat_ws(",", collect_set(a.address)) as addr
  > from t11_user u join t11_address a on u.id = a.uid
  > group by u.id;
Automatically selecting local only mode for query
Query ID = crxy_20151230230808_3e8a8606-6d57-4703-bcbf-dd116d3a3e23
Total jobs = 1
```

第一步：将数据分割开来使用 split 函数

```
hive> select split(addr, ",") from t12_user_addr;
OK
_c0
["shangdong","gansu"]
["henan"]
["beijing"]
Time taken: 0.183 seconds, Fetched: 3 row(s)

Time taken: 0.100 seconds, Fetched: 3 row(s)
hive> select explode(split(addr)) from t12_user_addr;
FAILED: SemanticException [Error 10015]: Line 1:15 Argument
hive> select explode(split(addr, ",")) from t12_user_addr;
Automatically selecting local only mode for query
Query ID = crxy_20151230231212_e9d6b2f0-03ec-45f7-a8bd-ae34
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce
Job running in-process (local Hadoop)
Hadoop job information for Stage-1: number of mappers: 0; n
2015-12-30 23:12:21,297 Stage-1 map = 100%, reduce = 0%
Ended Job = job_local1634001860_0011
MapReduce Jobs Launched:
Stage-Stage-1: HDFS Read: 1849 HDFS Write: 1175 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
col
shangdong
gansu
henan
beijing
min +-----+-----+
max +-----+-----+
```

整体执行字段

```
hive> select id, name, explode(split(addr, ",")) from t12_user_addr;
FAILED: SemanticException [Error 10081]: UDTF's are not supported outside the SELECT clause, nor nested in expressions
hive>
```

在将 explode 和 split 等这些内嵌函数组合起来使用的时候，是不能直接放在查询字段的后面的，这里需要使用到关键字 **lateral view**，从另外一个角度来获取数据。

lateral view 用于和 split, explode 等 UDTF 一起使用，它能够将一行数据拆成多行数据，在此基础上可以对拆分后的数据进行聚合。lateral view 首先为原始表的每行调用 UDTF，UDTF 会把一行拆分成一或者多行，lateral view 再把结果组合，产生一个支持别名表的虚拟表。

```
select id, name, address from t12_user_addr lateral view explode(split(addr,
```

```
" , " )) a as address;

hive> select id, name, address from t12_user_addr lateral view explode(split(addr, ",")) a as address;
Automatically selecting local only mode for query
Query ID = crxy_20151230231616_89f5165a-ca34-46fa-a1c7-f6e9218a2a0d
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Job running in-process (local Hadoop)
Hadoop job information for Stage-1: number of mappers: 0; number of reducers: 0
2015-12-30 23:16:08,989 Stage-1 map = 100%,  reduce = 0%
Ended Job = job_local1615451096_0012
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 1936 HDFS Write: 1243 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
id      name      address
1      zhangsan    shangdong
1      zhangsan    gansu
2      lisi        henan
3      wangwu     beijing
Time taken: 3.014 seconds. Fetched: 4 row(s)
```

5.2.3. Hive 自定义函数

那有时候，这些 hive 内嵌的函数，虽然说功能非常的强大，但是我们的业务可能是千变万化的，他有时候还是不能很好的满足我们的一些个性化的要求，而我们又想完成我们的工作，那么该怎么办呢？这个时候呢，就可以通过我们自定义的函数来完成我们的功能需求了，这个时候就需要我们写一点 java 代码了，步骤如下：

- 1°、自定义 UDF extends org.apache.hadoop.hive.ql.exec.UDF
- 2°、需要实现 **evaluate** 函数，evaluate 函数支持重载
- 3°、把程序打包放到目标机器上去
- 4°、进入 hive 客户端，添加 jar 包：hive>add jar jar 路径
- 5°、创建临时函数：hive> CREATE TEMPORARY FUNCTION 自定义名称 AS '自定义 UDF 的全类名'
- 6°、执行 HQL 语句；
- 7°、销毁临时函数：hive> DROP TEMPORARY FUNCTION 自定义名称

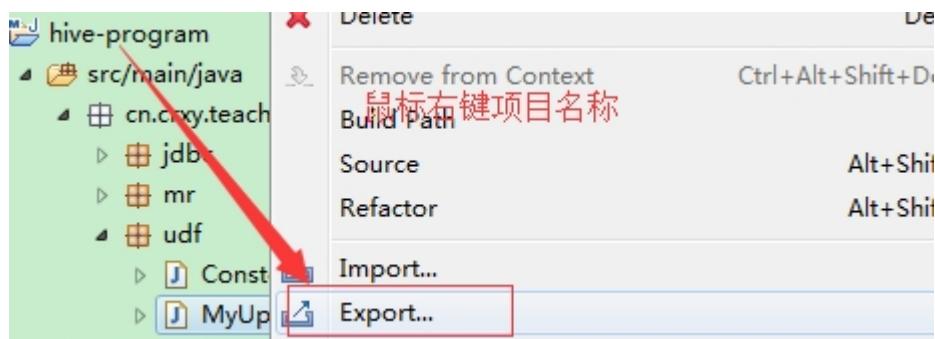
注：UDF 只能实现一进一出的操作，如果需要实现多进一出，则需要实现 UDAF（聚合函数），做法类似

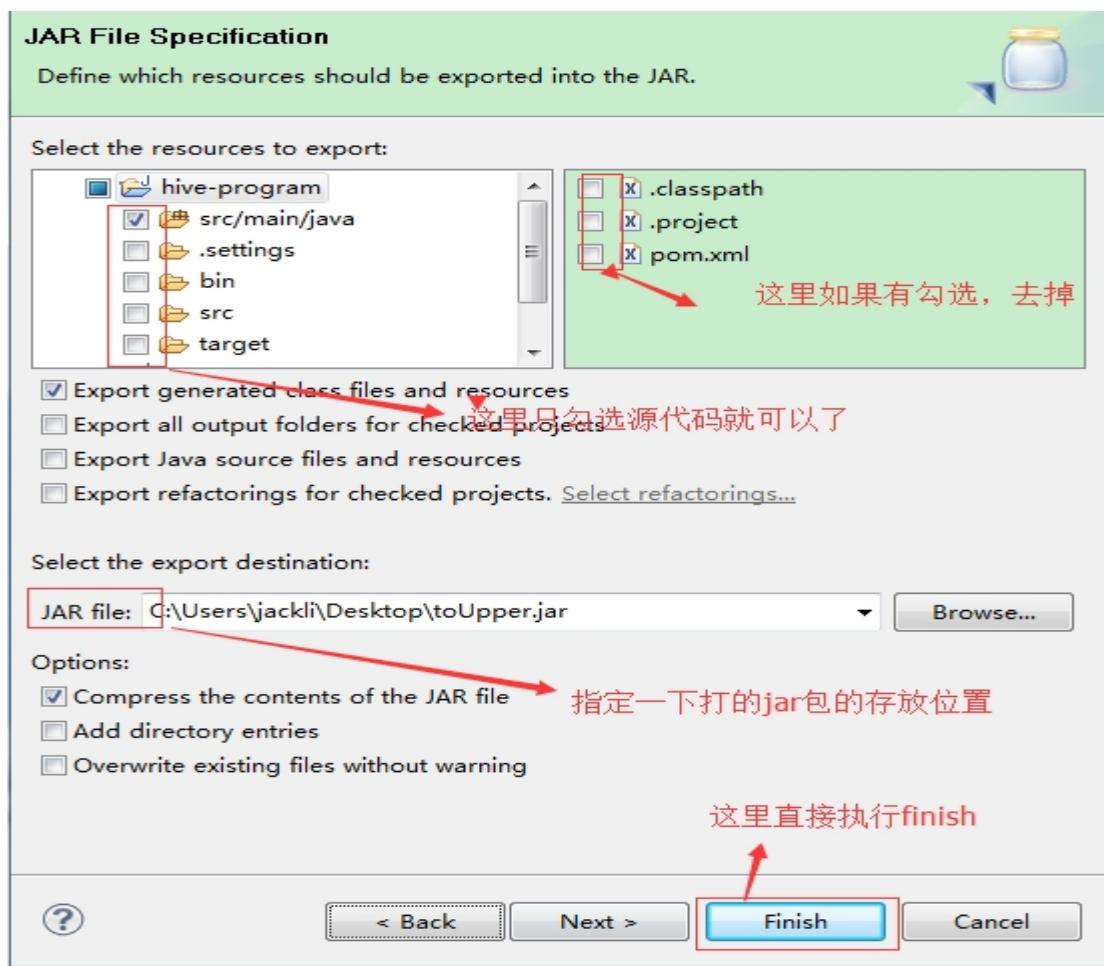
eg. 这里我们以仿写 UDF 中 upper (字符变大写) 函数为例

- 1°、自定义 UDF extends org.apache.hadoop.hive.ql.exec.UDF
- 2°、需要实现 **evaluate** 函数，evaluate 函数支持重载

```
package cn.crxy.teach.hive.udf;
import org.apache.hadoop.hive.ql.exec.Description;
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;
/*下面的注解是对函数的注释 可以通过 desc function fName 获取*/
@Description(name = "MyUpper",
    value = " _FUNC_(str) - Returns str with all characters changed
to uppercase",
    extended = "Example:\n"
    + " > SELECT _FUNC_(name) FROM src;")
public class MyUpper extends UDF {
    public Text evaluate(Text text) {
        if(text != null) {
            return new Text(text.toString().toUpperCase());
        } else {
            return null;
        }
    }
}
```

3°、把程序打包放到目标机器上去





4°、进入 hive 客户端，添加 jar 包: hive>add jar jar 路径

```
hive> add jar data/toUpperCase.jar
      ;
Added [data/toUpperCase.jar] to class path
Added resources: [data/toUpperCase.jar]
```

5°、创建临时函数: hive> CREATE TEMPORARY FUNCTION 自定义名称 AS '自定义 UDF 的全类名'

```
hive> create temporary function toUpper as 'cn.crxy.teach.hive.udf.MyUpper';
OK
Time taken: 0.103 seconds
```

```
hive> desc function toUpper;
OK
tab_name                               我们自定义的
toUpper(str) - Returns str with all characters changed to uppercase
Time taken: 0.063 seconds, Fetched: 1 row(s)
hive> desc function upper;
OK
tab_name                               系统自带的
upper(str) - Returns str with all characters changed to uppercase
Time taken: 0.1 seconds, Fetched: 1 row(s)
```

6°、执行 HQL 语句

```
hive> select id, stu_name, upper(stu_name) as UName, toUpper(stu_name) as tUName from t2;
OK
id      stu_name      uname    tuname
1      zhangsan      ZHANGSAN      ZHANGSAN          效果和系统自带的一致
NULL    NULL        NULL      NULL
NULL    NULL        NULL      NULL
Time taken: 0.422 seconds, Fetched: 3 row(s)
```

7°、销毁临时函数: hive> DROP TEMPORARY FUNCTION 自定义名称

如果只是作为临时使用，以后不用了的话，在执行完当次任务之后，就可以销毁了

```
Time taken: 0.053 seconds
hive> drop temporary function toUpper;
OK
Time taken: 0.046 seconds
hive> desc function toUpper;          删除自建的UDF, not exist显示已经删除了～
OK
tab_name
Function 'toUpper' does not exist.
Time taken: 0.146 seconds, Fetched: 1 row(s)
hive> create temporary function toUpper1 as 'cn.crxy.teach.hive.udf.MyUpper';
OK
Time taken: 0.017 seconds
hive> desc function toUpper1;          jar还在classpath下面，还允许再一次创建临时函数
OK
tab_name
toUpper1(str) - Returns str with all characters changed to uppercase
Time taken: 0.062 seconds, Fetched: 1 row(s)
hive>
```

5.2.4. Hive JDBC

这里说最后一个知识点，也是前面提到的 Hive 的三种访问方式中的最后一种，使用 jdbc 的方式来访问 hive 数据库，其实非常的简单，就把 hive 当做 mysql 来对待就 ok 了，mysql 怎么建立 jdbc，hive 就怎么建立 jdbc，一样一样的。

注意：在使用 jdbc 的方式之前呢，必须先启动 hive jdbc 端口

```
[crxy@master data]$ sh ../deployed-soft/hive-0.14.0/bin/hive --service hiveserver -p 10000
Starting Hive Thrift Server
```

这是前台进程，我们可以将其转换为后台进程，不用一直站着前端界面，我们无法使用当前窗口，执行下面命令即可

```
[root@hive local] nohup sh /usr/local/hive-0.14.0/bin/hive --service
hiveserver -p 10000 >/dev/null 2>&1 & (将产生的日志写到回收站里面)

[root@service ~]# nohup sh /usr/local/hive-0.14.0/bin/hive --service hiveserver -p 10000 >/dev/null 2>&1 &
```

如果不用的话，也可将该进程杀掉

```
[root@service local]# netstat -tunlp | grep 10000
tcp        0      0 0.0.0.0:10000          0.0.0.0:*                  LISTEN      29401/java
[root@service local]# kill -9 29401           找到10000端口对应的进程id (pid)，然后杀死
[root@service local]# netstat -tunlp | grep 10000
[root@service local]#
```

jdbc 代码编写

```
package cn.crxy.teach.hive.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class HiveJdbc {

    public static void main(String[] args) throws Exception {
        Class.forName("org.apache.hadoop.hive.jdbc.HiveDriver");
        Connection con = DriverManager.

        getConnection("jdbc:hive://hive.teach.crxy.cn:10000/default");
        PreparedStatement ps = con.
            prepareStatement("select wc.word, count(1) as
count " +
                "from (select explode(split(word, ' ')) as word
from t10_wc) wc " +
                "group by wc.word order by count desc");
        ResultSet rs = ps.executeQuery();
        while(rs.next()) {
            String word = rs.getString(1);
            int count = rs.getInt(2);
            System.out.println(word + ", " + count);
        }
        rs.close();
        ps.close();
        con.close();
    }
}
```

pom.xml 文件

```
<properties>
    <hive-api.version>0.14.0</hive-api.version>
    <hadoop-api.version>2.6.0</hadoop-api.version>
    <hadoop-core.version>1.2.1</hadoop-core.version>
</properties>
<dependencies>
    <dependency>
        <groupId>net.hydromatic</groupId>
        <artifactId>linq4j</artifactId>
        <version>0.1.12-SNAPSHOT</version>
    </dependency>
```

```
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <version>${hadoop-api.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-mapreduce-client-core</artifactId>
    <version>${hadoop-api.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-core</artifactId>
    <version>${hadoop-core.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-exec</artifactId>
    <version>${hive-api.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-serde</artifactId>
    <version>${hive-api.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-service</artifactId>
    <version>${hive-api.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-metastore</artifactId>
    <version>${hive-api.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-common</artifactId>
    <version>${hive-api.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-cli</artifactId>
    <version>${hive-api.version}</version>

```

```

</dependency>
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-jdbc</artifactId>
    <version>${hive-api.version}</version>
</dependency>
<dependency>
    <groupId>org.apache.thrift</groupId>
    <artifactId>libfb303</artifactId>
    <version>0.9.0</version>
</dependency>
</dependencies>

```

运行结果:

```

hello, 3
jack, 1
dachui, 1
bella, 1

```

在 hive 后台显示，执行完 mr 了

```

MapReduce Total cumulative CPU time: 7 seconds 640 msec
Ended Job = job_1451357225044_0015
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1  Cumulative CPU: 8.62 sec   HDFS Read: 256 HDFS Write: 192 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1  Cumulative CPU: 7.64 sec   HDFS Read: 604 HDFS Write: 32 SUCCESS
Total MapReduce CPU Time Spent: 16 seconds 260 msec
OK

```

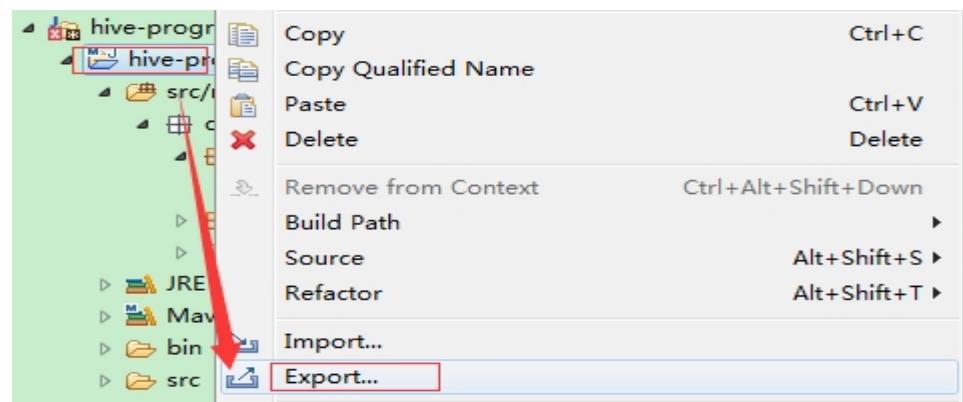
说明：在 windows 本地执行的时候可能会报错，报错不过用管，对我们的执行没有影响

```

16/01/11 15:18:43 ERROR util.Shell: Failed to locate the winutils binary in the hadoop binary path
java.io.IOException: Could not locate executable null\bin\winutils.exe in the Hadoop binaries.
    at org.apache.hadoop.util.Shell.getQualifiedBinPath(Shell.java:355)
    at org.apache.hadoop.util.Shell.getWinUtilsPath(Shell.java:370)
    at org.apache.hadoop.util.Shell.<clinit>(

```

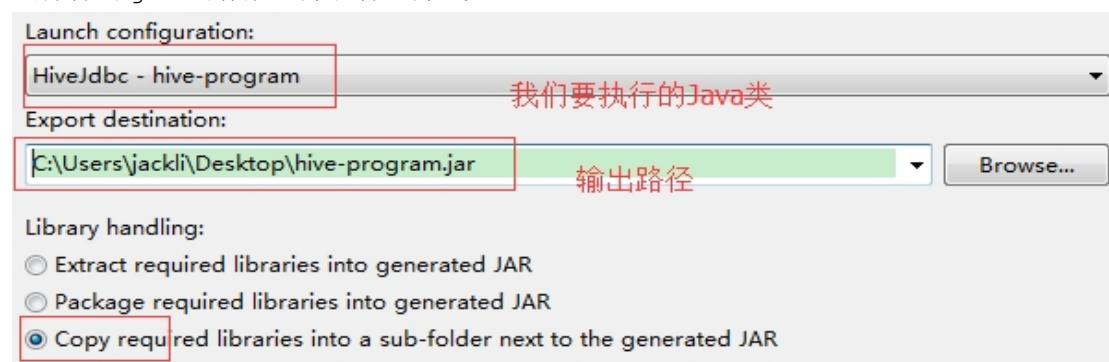
是因为 window 本地缺少非常多的配置信息，这里就不做过多说明，把我们写完的 jdbc 程序，部署到 linux 下面跑是正常没有问题的！



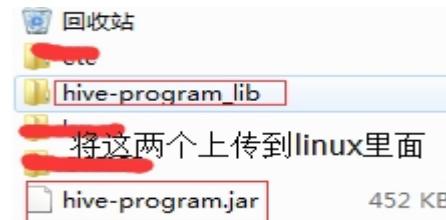
选择 java/runnable jar file



选择打的 jar 的存放地方和打包方式



将打好的 jar 包上传至 linux 服务器



```
[root@service data]# pwd
/usr/local/data
[root@service data]# ll
total 464
-rw-r--r--. 1 root root 462585 Jan 10 23:16 hive-program.jar
drwxr-xr-x. 2 root root    4096 Jan 10 23:16 hive-program_lib
-rw-r--r--. 1 root root      10 Jan 10 22:26 t1
```

然后执行: `java -jar hive-program.jar cn.crxy.teach.hive.jdbc.HiveJdbc`

```
[root@service data]# java -jar hive-program.jar cn.crxy.teach.hive.jdbc.HiveJdbc
hello, 3
you, 1
me, 1
jack, 1
```

它是没有报错的，所以说在 windows 下的报错不用管

5.3. 考题

5.4. 实验

附件说明：

1、将普通用户赋予 sudo 的权限

1°、将用户切换到 root 用户下

```
[crxy@master ~]$ su -l
Password:
[root@master ~]#
```

2°、查看并修改/etc/sudoers 文件权限

```
[root@master ~]# ll /etc/sudoers
-r--r-----. 1 root root 4022 Jan  8 22:44 /etc/sudoers
```

发现它是没有写的权限的，需要我们增加写的权限

```
[root@master ~]# chmod u+w /etc/sudoers
[root@master ~]# ll /etc/sudoers
-rw-r-----. 1 root root 4022 Jan  8 22:44 /etc/sudoers
[root@master ~]#
```

3°、在文件中增加我们普通用户，让普通用也可以执行一些命令

vi /etc/sudoers

大概在如下图所示的位置处，加一行和 98 相同的内容，不同的是把，第一列 root 改成 crxy 我们的普通用户，然后保存退出。

```
97 ## Allow root to run any commands anywhere
98 root    ALL=(ALL)      ALL
99 crxy   ALL=(ALL)      ALL
```

最简单的把98行的内容拷贝一份，
把第一个root改成自己的普通用户

4°、权限恢复原状

将我们修改的/etc/sudoers 权限恢复原状，不然可能会有这样那样的问题!!!

```
[root@master ~]# chmod u-w /etc/sudoers
[root@master ~]# ll /etc/sudoers
-r--r-----. 1 root root 4023 Jan 10 10:35 /etc/sudoers
```

这个时候再使用 sudo 就可以不会报错误了

```
[crxy@master ~]$ sudo rpm -qa | grep MySQL
[sudo] password for crxy:
[crxy@master ~]$
```

详细的过程呢，大家也可以参考这个网址：

<http://www.cnblogs.com/zox2011/archive/2013/05/28/3103824.html>

2、MySQL 编码设置

普通情况下咱们的 mysql 默认编码是 latin1，但是在我们的日常开发中大都情况下需要用到 utf8 编码，如果是默认 latin1 的话，咱们的中文存储进去容易乱码的，所以说如果大家在遇到一些数据库乱码的话，最好把 mysql 的编码改成 utf8.

但是在这里非常严重的强调一点，hive 的元数据 metastore 在 mysql 的数据库，不管是数据库本身，还是里面的表，抑或表中的字段的编码都必须是 latin1 (CHARACTER SET latin1 COLLATE latin1 bin)!!!

不然会有类似如下的错误!!!

```
创建表报错
NestedThrowablesStackTrace:
com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Specified key was too long; max key length is 767 bytes
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
错误是因为hive的存储引擎的编码为utf8操作, [hive的存储引擎必须为latin1]
```

那么怎么修改 mysql 的编码为 utf8 呢，这里提供了在线方式安装和离线方式安装下的修改方式供大家选择！

2.1、离线安装 mysql 的修改方式

修改编码，设置为 utf8

拷贝 mysql 的配置文件 /usr/share/mysql/my-small.cnf 到 /etc/my.cnf

```
[crxy@master soft]$ sudo cp /usr/share/mysql/my-small.cnf /etc/my.cnf
```

在 mysql 配置文件 /etc/my.cnf 中增加以下内容

在 [client] 下面增加

```
default-character-set=utf8
```

在 [mysqld] 下面增加

```
default-character-set=utf8
```

init_connect='SET NAMES utf8' 如下图

```
[client]
#password      = your_password
port          = 3306
socket        = /var/lib/mysql/mysql.sock
default-character-set=utf8
# Here follows entries for some specific programs

# The MySQL server
[mysqld]
port          = 3306
socket        = /var/lib/mysql/mysql.sock
skip-locking
key_buffer_size = 16K           括起来的是新增内容
max_allowed_packet = 1M
table_open_cache = 4
sort_buffer_size = 64K
read_buffer_size = 256K
read_rnd_buffer_size = 256K
net_buffer_length = 2K
thread_stack = 128K
default-character-set=utf8
init_connect='SET NAMES utf8'
```

第四步：重启 mysql 服务

```
sudo service mysql restart
```

```
[crxy@master soft]$ sudo service mysql restart
Shutting down MySQL... SUCCESS!
Starting MySQL. SUCCESS!
```

第五步：编码验证：在终端输入 show variables like 'char%';

```
mysql> show variables like 'char%';
+-----+-----+
| Variable_name      | Value
+-----+-----+
| character_set_client | utf8
| character_set_connection | utf8
| character_set_database | utf8
| character_set_filesystem | binary
| character_set_results | utf8
| character_set_server | utf8
| character_set_system | utf8
| character_sets_dir   | /usr/share/mysql/charsets/
+-----+-----+
8 rows in set (0.00 sec)
```

OK，修改成功！

2.2、在线安装 mysql 的修改方式

修改编码，设置为 utf-8

在 mysql 配置文件/etc/my.cnf（不需要拷贝）中 [mysqld] 的下面增加以下内容

```
[crxy@hive ~]$ sudo vi /etc/my.cnf
[sudo] password for crxy:

[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted security risks
symbolic-links=0

init_connect='SET collation_connection = utf8_unicode_ci'
init_connect='SET NAMES utf8'
character-set-server=utf8
collation-server=utf8_unicode_ci
skip-character-set-client-handshake

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
*
```

新增的内容

```
init_connect='SET collation_connection = utf8_unicode_ci'
init_connect='SET NAMES utf8'
character-set-server=utf8
collation-server=utf8_unicode_ci
skip-character-set-client-handshake
```

第四步：重启 mysqld 服务

```
sudo service mysqld restart
```

```
[crxy@master ~]$ sudo service mysqld restart
Stopping mysqld: [OK]
Starting mysqld: [OK]
```

第五步：编码验证，在终端输入 show variables like 'char%';

```
mysql> show variables like 'char%';
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| character_set_client | utf8    |
| character_set_connection | utf8    |
| character_set_database | utf8    |
| character_set_filesystem | binary |
| character_set_results | utf8    |
| character_set_server | utf8    |
| character_set_system | utf8    |
| character_sets_dir   | /usr/share/mysql/charsets/ |
+-----+-----+
8 rows in set (0.00 sec)
```

OK，修改成功！

3、Hive 参数说明

这里提供一个 blog 地址，大家可以在[这里](http://blog.csdn.net/w13770269691/article/details/17232947)查看详细的 Hive 配置参数的中文解释

<http://blog.csdn.net/w13770269691/article/details/17232947>

4、字段定义时的 Comment

在创建表的时候，字段可以有 comment，但是 comment 建议不要用中文说明，因为我们说过，hive 的 metastore 支持的字符集是 latin1，所以中文写入的时候会有编码问题，如下图！

```

hive> create table t2(
>   id int comment 'id',
>   name string comment '名称'
> );
OK
Time taken: 0.084 seconds
hive> desc t2;
OK
+-----+-----+-----+
| col_name | data_type | comment |
+-----+-----+-----+
| id       | int      |          |
| name     | string   |          |
+-----+-----+-----+
Time taken: 0.189 seconds, Fetched: 2 row(s)
  
```

col_name	data_type	comment	
id	int		id
name	string	名称	??

5、关于 Sequence 不支持 GzipCodec

```

... 9 more
Caused by: org.apache.hadoop.hive.ql.metadata.HiveException: java.lang.IllegalArgumentException: SequenceFile
doesn't work with GzipCodec without native-hadoop code!
    at org.apache.hadoop.hive.ql.io.HiveFileFormatUtils.getHiveRecordWriter(HiveFileFormatUtils.java:277)
    at org.apache.hadoop.hive.ql.exec.FileSinkOperator.createBucketForFileIdx(FileSinkOperator.java:562)
    at org.apache.hadoop.hive.ql.exec.FileSinkOperator.createBucketFiles(FileSinkOperator.java:506)
    ... 16 more
Caused by: java.lang.IllegalArgumentException: SequenceFile doesn't work with GzipCodec without native-hadoop
code!
    at org.apache.hadoop.io.SequenceFile$Writer.<init>(SequenceFile.java:1088)
  
```

这个时候我们来用命令 `hadoop checknative -a` 检测一下可以使用的压缩依赖 lib 库，结果发现，都不支持，令人非常沮丧!!!

```

[root@service local]# hadoop checknative -a
16/01/11 22:45:14 WARN util.NativeCodeLoader: Unable
Native library checking:
hadoop: false
zlib: false
snappy: false
lz4: false
bzip2: false
openssl: false
  
```

这里因为使用的是 32bit 的 hadoop 版本，如果不对源码编译的话是不会支持这种压缩的，所以需要对源码进行编译，使用 64bit 版本的 hadoop，或者更为简单的做法是讲编译好的相同版本的 hadoop 的 lib/native 直接拷贝到原来 32bit 的 lib/native 目录下面。

```
[root@service lib]# cp -r native/ native-bak
[root@service lib]# ll
total 8
drwxr-xr-x. 2 root root 4096 Jan 10 00:22 native
drwxr-xr-x. 2 root root 4096 Jan 11 22:46 native-bak
[root@service lib]# cp /usr/local/hadoop-2.6.0-64bit/lib/native/* native/
cp: overwrite `native/libhadoop.a'? y
cp: overwrite `native/libhadooppipes.a'? y
cp: overwrite `native/libhadoop.so'? y
cp: overwrite `native/libhadoop.so.1.0.0'? y
cp: overwrite `native/libhadooputils.a'? y
cp: overwrite `native/libhdfs.a'? y
cp: overwrite `native/libhdfs.so'? y
cp: overwrite `native/libhdfs.so.0.0.0'? y
```

再检测，有些选项没有，那也没有办法，但是目前没有发现对我们的使用有影响。

```
[root@service lib]# hadoop checknative -a
16/01/11 22:47:32 WARN bzip2.Bzip2Factory: Failed to load/initialize native-bzip2
16/01/11 22:47:32 INFO zlib.ZlibFactory: Successfully loaded & initialized native
Native library checking:
hadoop: true /usr/local/hadoop-2.6.0/lib/native/libhadoop.so
zlib: true /lib64/libz.so.1
snappy: false
lz4: true revision:99
bzip2: false
openssl: false Cannot load libcrypto.so (libcrypto.so: cannot open shared object
16/01/11 22:47:32 INFO util.ExitUtil: Exiting with status 1
```

再来执行数据插入，OK！

```
hive> insert into table test_sequence select * from test;
Query ID = root_20160111225252_20700610-e41d-499e-a39f-e28a4cc99a25
Total jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1452581068794_0001, Tracking URL = http://service.base.crxy.cn:808
581068794_0001/
Kill Command = /usr/local/hadoop-2.6.0/bin/hadoop job -kill job_1452581068794_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2016-01-11 22:53:01,585 Stage-1 map = 0%, reduce = 0%
2016-01-11 22:53:17,689 Stage-1 map = 70%, reduce = 0%, cumulative CPU 12.2 sec
2016-01-11 22:53:18,742 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 13.31 sec
MapReduce Total cumulative CPU time: 13 seconds 310 msec
Ended Job = job_1452581068794_0001
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: hdfs://service.base.crxy.cn:9000/tmp/hive/root/9ddfa410-6415-4274-99c
1/-ext-10000
Loading data to table default.test_sequence
Table default.test_sequence stats: [numFiles=1, numRows=2400004, totalSize=935880, ra
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1   Cumulative CPU: 13.31 sec   HDFS Read: 192004711 HDFS Write:
Total MapReduce CPU Time Spent: 13 seconds 310 msec
OK
Time taken: 34.749 seconds
```

而且呢，咱们之前的 32bit 的 hadoop 也就华丽转化为 64bit 的啦！

```
[root@service native]# pwd  
/usr/local/hadoop-2.6.0/lib/native  
[root@service native]# file libhadoop.so.1.0.0  
libhadoop.so.1.0.0: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked  
[root@service native]#
```

我们在 hadoop 开发中实际上常用的压缩方式有四种, bzip2, gzip, snappy, lzo,但是 hadoop 只默认支持 gzip 和 bzip2, 其他两种压缩方式需要我们手动来安装。

下面给大家介绍几篇 blog 来说说四种的压缩方式和压缩方式的安装配置, 下来可以参考一下。

第一篇: [hadoop 中 4 种压缩格式的特征的比较](#)

第二篇: [Hadoop 安装配置 snappy 压缩](#)

第三篇: [Hadoop 压缩算法 snappy 和 lzo 的安装](#)