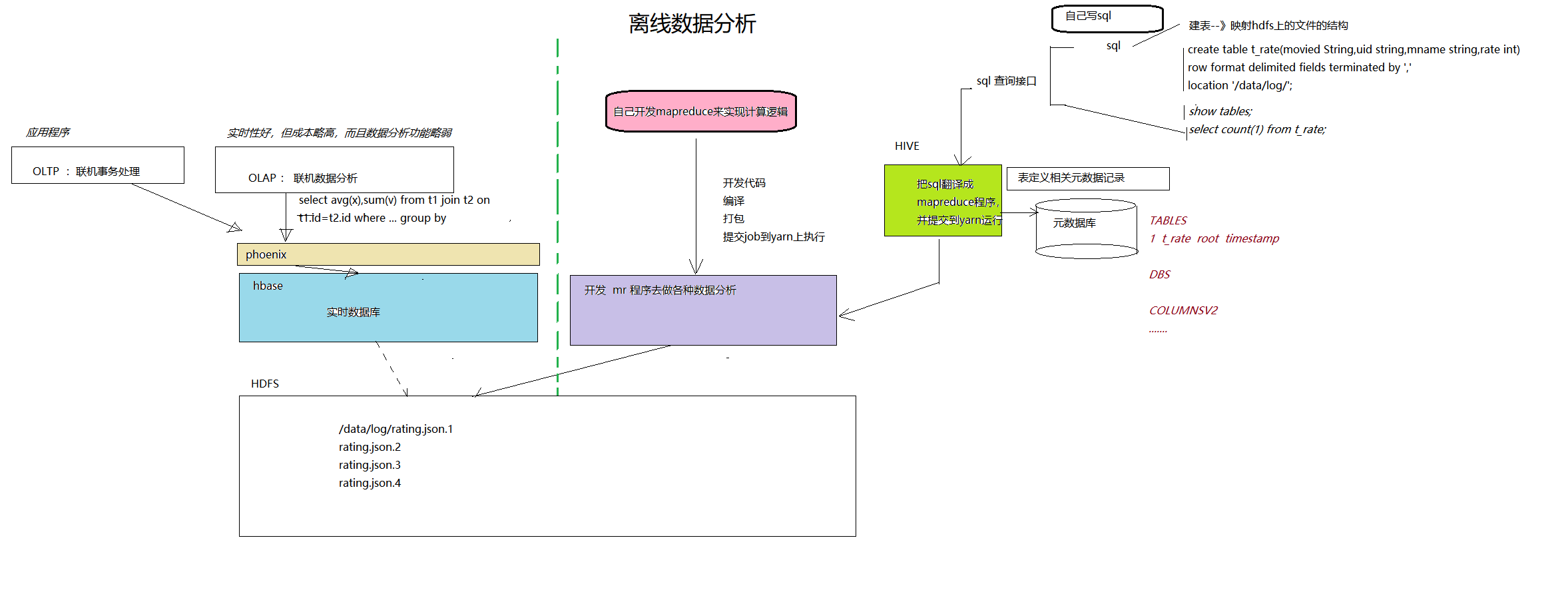


《多易教育：HIVE教程》

# hive入门

## 数据分析概念



## 什么是Hive

Hive：由Facebook开源用于解决海量结构化日志的数据统计。

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

本质是：将HQL转化成MapReduce程序



1）Hive处理的数据存储在HDFS

2）Hive分析数据底层的实现是MapReduce / spark(分布式运算框架)

3）执行程序运行在Yarn上

[运行HDFS和YARN ]

## Hive的优缺点

### 优点

1. 操作接口采用类SQL语法，提供快速开发的能力（简单、容易上手）。
2. 避免了去写MapReduce，减少开发人员的学习成本。
3. Hive的执行延迟比较高，因此Hive常用于数据分析，对实时性要求不高的场合。(历史数据)
4. Hive优势在于处理大数据，对于处理小数据没有优势，因为Hive的执行延迟比较高。
5. Hive支持用户自**定义函数**，用户可以根据自己的需求来实现自己的函数。

### 缺点

sql hive sql hql

1．Hive的HQL表达能力有限

（1）迭代式算法无法表达

（2）数据挖掘方面不擅长

2．Hive的效率比较低

（1）Hive自动生成的MapReduce作业，通常情况下不够智能化

（2）Hive调优比较困难，粒度较粗(快)

## Hive架构原理

(mysql记录元数据)



1．用户接口：Client

CLI（hive shell）、JDBC/ODBC(java访问hive)、WEBUI（浏览器访问hive）

2．元数据：Metastore

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

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

3．Hadoop

使用HDFS进行存储，使用MapReduce进行计算。

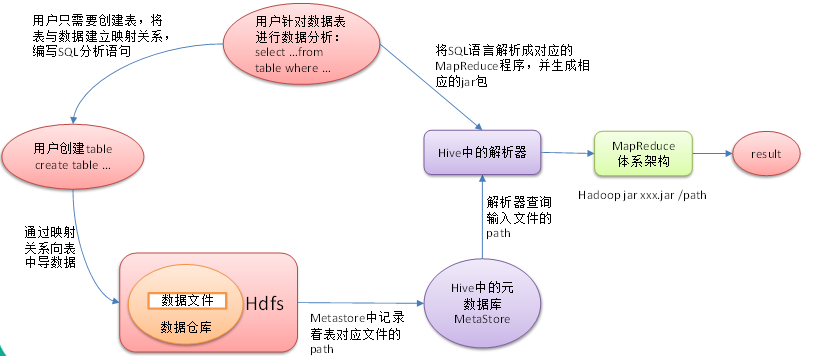
4．驱动器：Driver

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

（2）编译器（Physical Plan）：将AST编译生成逻辑执行计划。

（3）优化器（Query Optimizer）：对逻辑执行计划进行优化。

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



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

## Hive和数据库比较

由于 Hive 采用了类似SQL 的查询语言 HQL(Hive Query Language)，因此很容易将 Hive 理解为数据库。其实从结构上来看，Hive 和数据库除了拥有类似的查询语言，再无类似之处。

本文将从多个方面来阐述 Hive 和数据库的差异。数据库可以用在 Online 的应用中，但是Hive 是为数据仓库而设计的，清楚这一点，有助于从应用角度理解 Hive 的特性。

### 查询语言

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

### 数据存储位置

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

### 数据更新

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

### 索引

Hive在加载数据的过程中不会对数据进行任何处理，甚至不会对数据进行扫描，因此也没有对数据中的某些Key建立索引。Hive要访问数据中满足条件的特定值时，需要暴力扫描整个数据，因此访问延迟较高。由于 MapReduce 的引入， Hive 可以并行访问数据，因此即使没有索引，对于[大数据](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "_blank)量的访问，Hive 仍然可以体现出优势。

数据库中，通常会针对一个或者几个列建立索引，因此对于少量的特定条件的数据的访问，数据库可以有很高的效率，较低的延迟。由于数据的访问延迟较高，决定了 Hive 不适合在线数据查询。

### 执行

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

### 执行延迟

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

### 可扩展性

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

### 数据规模

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

# hive的安装

## Hive安装地址

1．Hive官网地址

http://hive.apache.org/

2．文档查看地址

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

3．下载地址

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

4．github地址

https://github.com/apache/hive

## Hive安装部署

1．Hive安装及配置

（1）把apache-hive-2.3.5-bin.tar.gz上传到linux的/usr/apps目录下

（2）解压apache-hive-2.3.5-bin.tar.gz到/usr/apps目录下面

[root@doit01 apps]$ tar -zxvf apache-hive-2.3.5-bin.tar.gz

（3）修改apache-hive-2.3.5-bin.tar.gz的名称为hive

[root@doit01 apps]$ mv apache-hive-2.3.5-bin/ hive-2.3.5

（4）修改/opt/module/hive/conf目录下的hive-env.sh.template名称为hive-env.sh

[root@doit01 conf]$ mv hive-env.sh.template hive-env.sh

（5）配置hive-env.sh文件

（a）配置HADOOP\_HOME路径

export HADOOP\_HOME=/usr/apps/hadoop-2.8.5

（b）配置HIVE\_CONF\_DIR路径

export HIVE\_CONF\_DIR=/usr/apps/hive-2.3.5/conf

2．Hadoop集群配置

（1）必须启动hdfs和yarn

[root@doit01 hadoop-2.8.5]$ sbin/start-dfs.sh

[root@doit01 hadoop-2.8.5]$ sbin/start-yarn.sh

（2）在HDFS上创建/tmp和/user/hive/warehouse两个目录并修改他们的同组权限可写

[root@doit01 hadoop-2.7.2]$ bin/hadoop fs -mkdir /tmp

[root@doit01 hadoop-2.7.2]$ bin/hadoop fs -mkdir -p /user/hive/warehouse

[root@doit01 hadoop-2.7.2]$ bin/hadoop fs -chmod g+w /tmp

[root@doit01 hadoop-2.7.2]$ bin/hadoop fs -chmod g+w /user/hive/warehouse

3．Hive基本操作

（1）启动hive

[root@doit01 hive]$ bin/hive

（2）查看数据库

hive> show databases;

（3）打开默认数据库

hive> use default;

（4）显示default数据库中的表

hive> show tables;

（5）创建一张表

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

（6）显示数据库中有几张表

hive> show tables;

（7）查看表的结构

hive> desc student;

（8）向表中插入数据

hive> insert into student values(1000,"ss");

（9）查询表中数据

hive> select \* from student;

（10）退出hive

hive> quit;

## 将本地文件导入Hive案例

需求

将本地/opt/module/datas/student.txt这个目录下的数据导入到hive的student(id int, name string)表中。

1．数据准备

在/opt/module/datas这个目录下准备数据

（1）在/opt/module/目录下创建datas

[root@doit01 module]$ mkdir datas

（2）在/opt/module/datas/目录下创建student.txt文件并添加数据

[root@doit0 1datas]$ touch student.txt

[root@doit01 datas]$ vi student.txt

1001 zhangshan

1002 lishi

1003 zhaoliu

注意以tab键间隔。

2．Hive实际操作

（1）启动hive

[root@doit01 hive]$ bin/hive

（2）显示数据库

hive> show databases;

（3）使用default数据库

hive> use default;

（4）显示default数据库中的表

hive> show tables;

（5）删除已创建的student表

hive> drop table student;

（6）创建student表, 并声明文件分隔符’\t’

hive> create table student(id int, name string) ROW FORMAT DELIMITED FIELDS TERMINATED

BY '\t';

（7）加载/opt/module/datas/student.txt 文件到student数据库表中。

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

（8）Hive查询结果

hive> select \* from student;

OK

1001 zhangshan

1002 lishi

1003 zhaoliu

Time taken: 0.266 seconds, Fetched: 3 row(s)

3．遇到的问题

再打开一个客户端窗口启动hive，会产生java.sql.SQLException异常。

Exception in thread "main" java.lang.RuntimeException: java.lang.RuntimeException:

Unable to instantiate

org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClient

at org.apache.hadoop.hive.ql.session.SessionState.start(SessionState.java:522)

at org.apache.hadoop.hive.cli.CliDriver.run(CliDriver.java:677)

at org.apache.hadoop.hive.cli.CliDriver.main(CliDriver.java:621)

at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)

at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)

at java.lang.reflect.Method.invoke(Method.java:606)

at org.apache.hadoop.util.RunJar.run(RunJar.java:221)

at org.apache.hadoop.util.RunJar.main(RunJar.java:136)

Caused by: java.lang.RuntimeException: Unable to instantiate org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClient

at org.apache.hadoop.hive.metastore.MetaStoreUtils.newInstance(MetaStoreUtils.java:1523)

at org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.<init>(RetryingMetaStoreClient.java:86)

at org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.getProxy(RetryingMetaStoreClient.java:132)

at org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.getProxy(RetryingMetaStoreClient.java:104)

at org.apache.hadoop.hive.ql.metadata.Hive.createMetaStoreClient(Hive.java:3005)

at org.apache.hadoop.hive.ql.metadata.Hive.getMSC(Hive.java:3024)

at org.apache.hadoop.hive.ql.session.SessionState.start(SessionState.java:503)

... 8 more

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

## MySql安装

### 移除原来的libs

防止冲突

 yum remove mysql-libs\*

### 安装顺序

rpm -ivh mysql-community-common-5.7.12-1.el6.x86\_64.rpm

rpm -ivh mysql-community-libs-5.7.12-1.el6.x86\_64.rpm

rpm -ivh mysql-community-client-5.7.12-1.el6.x86\_64.rpm

rpm -ivh mysql-community-server-5.7.12-1.el6.x86\_64.rpm

rpm -ivh mysql-community-devel-5.7.12-1.el6.x86\_64.rpm

yum -y install numactl.x86\_64

### 启动Mysql服务

# service mysqld start

初始化 MySQL 数据库：                                [确定]

Installing validate pass[word](http://www.itmop.com/key/word/" \t "http://www.itmop.com/downinfo/_blank) plugin:                       [确定]

正在启动 mysqld：

### 修改管理员密码

grep 'temporary password' /var/log/mysqld.log

mysql> set global validate\_password\_policy=0;

mysql> set global validate\_password\_length=4;

mysql> SET PASSWORD = PASSWORD('root');

### 设置远程登陆权限

给root用户授予从任何机器上登陆mysql服务器的权限：

mysql> **grant all privileges on \*.\* to 'root'@'%' identified by 'root' with grant option;**

**Query OK, 0 rows affected (0.00 sec)**

## Hive元数据配置到MySql

### 驱动拷贝

1．在/opt/software/mysql-libs目录下解压mysql-connector-java-5.1.27.tar.gz驱动包

[root@doit01 mysql-libs]# tar -zxvf mysql-connector-java-5.1.27.tar.gz

2.拷贝/opt/software/mysql-libs/mysql-connector-java-5.1.27目录下的mysql-connector-java-5.1.27-bin.jar到/opt/module/hive/lib/

[root@doit01 mysql-connector-java-5.1.27]# cp mysql-connector-java-5.1.27-bin.jar

/opt/module/hive/lib/

### 配置Metastore到MySql

1．在/opt/module/hive/conf目录下创建一个hive-site.xml

[root@doit01 conf]$ touch hive-site.xml

[root@doit01 conf]$ vi hive-site.xml

2．根据官方文档配置参数，拷贝数据到hive-site.xml文件中

<https://cwiki.apache.org/confluence/display/Hive/AdminManual+MetastoreAdmin>

hive的数据存储在HDFS上 , 在使用hive之前,hdfs先启动

hive的元数据信息建议存储在mysql数据库中, 安装hive之前建议先安装mysql存储元数据

cp hive-env.sh.template hive-env.sh

* vi hive-env.sh 自动的读取HDFS的配置

export HIVE\_CONF\_DIR=/usr/apps/hive/conf

export HADOOP\_HOME=/usr/apps/hadoop-2.8.5

* 在hive的conf中创建 hive-site.xml文件

<?xml version="1.0"?>

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

<configuration>

<property>

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

<value>jdbc:mysql://linux01:3306/metastore?createDatabaseIfNotExist=true&amp;useUnicode=true&characterEncoding=UTF-8</value>

<description>JDBC connect string for a JDBC metastore</description>

</property>

<property>

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

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

<description>Driver class name for a JDBC metastore</description>

</property>

<property>

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

<value>root</value>

<description>username to use against metastore database</description>

</property>

<property>

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

<value>root</value>

<description>password to use against metastore database</description>

</property>

</configuration>

### 多窗口启动Hive测试

1．先启动MySQL

[root@doit01 mysql-libs]$ mysql -uroot -p000000

查看有几个数据库

mysql> show databases;

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

| Database |

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

| information\_schema |

| mysql |

| performance\_schema |

| test |

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

2．再次打开多个窗口，分别启动hive

[root@doit01hive]$ bin/hive

3．启动hive后，回到MySQL窗口查看数据库，显示增加了metastore数据库

mysql> show databases;

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

| Database |

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

| information\_schema |

| metastore |

| mysql |

| performance\_schema |

| test |

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

## HiveJDBC访问

### 启动hiveserver2服务

[root@doit01 hive]$ bin/hiveserver2

### 启动beeline

[root@doit01 hive]$ bin/beeline

Beeline version 2.3.5 by Apache Hive

beeline>

### 连接hiveserver2

beeline> !connect jdbc:hive2:/doit01:10000（回车）

Connecting to jdbc:hive2:/doit01:10000

Enter username for jdbc:hive2:/doit01:10000: root（回车）

Enter password for jdbc:hive2:/doit01:10000: （直接回车）

Connected to: Apache Hive (version 2.3.5)

Driver: Hive JDBC (version 2.3.5)

Transaction isolation: TRANSACTION\_REPEATABLE\_READ

0: jdbc:hive2:/doit01:10000> show databases;

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

| database\_name |

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

| default |

| hive\_db2 |

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

## Hive常用交互命令

|  |
| --- |
| [root@doit01hive]$ bin/hive -help  usage: hive  -d,--define <key=value> Variable subsitution to apply to hive  commands. e.g. -d A=B or --define A=B  --database <databasename> Specify the database to use  -e <quoted-query-string> SQL from command line  -f <filename> SQL from files  -H,--help Print help information  --hiveconf <property=value> Use value for given property  --hivevar <key=value> Variable subsitution to apply to hive  commands. e.g. --hivevar A=B  -i <filename> Initialization SQL file  -S,--silent Silent mode in interactive shell  -v,--verbose Verbose mode (echo executed SQL to the console) |

1．“-e”不进入hive的交互窗口执行sql语句

[root@doit01hive]$ bin/hive -e "select id from student;"

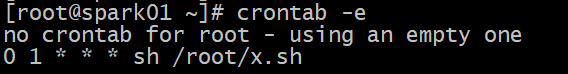
命令化执行sql的意义：可以将大量每天都要执行的sql，放入一个shell脚本，就可以每天自动化执行。

vi x.sh

|  |
| --- |
| #!/bin/bash  /opt/apps/hive-2.3.5/bin/hive -e "select max(id) from t1"  /opt/apps/hive-2.3.5/bin/hive -e "select count(1) from t1"  /opt/apps/hive-2.3.5/bin/hive -e "select id,name from t1 where id>3" |

然后，还可以把这个脚本的执行，配置到linux的定时任务中：

crontab -e

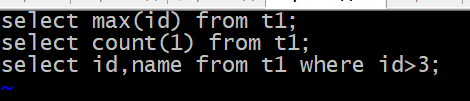


2．“-f”执行脚本中sql语句

（1）在/root目录下创建x.sql文件

[root@doit01datas]$ vi x.sql

文件中写入正确的sql语句



（2）执行文件中的sql语句

[root@doit01hive]$ bin/hive -f /root/x.sql

小提示：通过重定向可以把hive -f的结果写入文件中[可选]

[root@doit01hive]$ bin/hive -f /root/x.sql > /root/x\_result.txt

## Hive其他命令操作

1．退出hive交互窗口：

hive(default)>exit;

hive(default)>quit;

在新版的hive中没区别了，在以前的版本是有的：

exit:先隐性提交数据，再退出；

quit:不提交数据，退出；

2．在hive cli交互命令窗口中操作hdfs文件系统

hive(default)>dfs -ls /;

hive(default)>dfs -rm -r /abc;

3．在hive cli命令窗口中如何查看本地文件系统【beeline中不支持】

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

4．查看在hive交互程序中输入的所有历史命令

（1）进入到当前用户的根目录/root或/home/xuser

（2）查看. hivehistory文件

[root@doit01~]$ cat .hivehistory

## Hive常见属性配置

### Hive数据仓库位置配置

1）Default数据仓库的最原始位置是在hdfs上的：/user/hive/warehouse路径下。

2）在仓库目录下，没有对默认的数据库default创建文件夹。如果某张表属于default数据库，直接在数据仓库目录下创建一个文件夹。

3）修改default数据仓库原始位置（将hive-default.xml.template如下配置信息拷贝到hive-site.xml文件中）。

<property>

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

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

<description>location of default database for the warehouse</description>

</property>

<property>

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

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

<description>location of default database for the warehouse</description>

</property>

配置同组用户有执行权限

bin/hdfs dfs -chmod g+w /user/hive/warehouse

### 查询后信息显示配置

1）在hive-site.xml文件中添加如下配置信息，就可以实现显示当前数据库，以及查询表头信息配置。

<property>

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

<value>true</value>

</property>

<property>

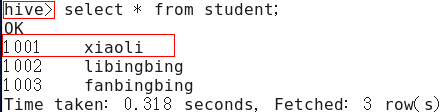
<name>hive.cli.print.current.db</name>

<value>true</value>

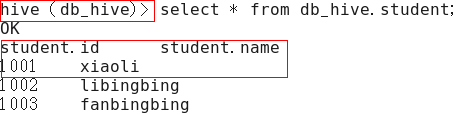
</property>

2）重新启动hive，对比配置前后差异。

（1）配置前，如图6-2所示



（2）配置后，如图6-3所示



### Hive运行日志存储目录配置

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

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

（1）修改/opt/module/hive/conf/hive-log4j.properties.template文件名称为hive-log4j.properties

[root@doit01conf]$ pwd

/opt/module/hive/conf

[root@doit01conf]$ mv hive-log4j.properties.template hive-log4j.properties

（2）在hive-log4j.properties文件中修改log存放位置

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

### 参数配置方式

1．查看当前所有的配置信息

hive>set;

2．参数的配置三种方式

**（1）配置文件方式**

默认配置文件：hive-default.xml

用户自定义配置文件：hive-site.xml

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

**（2）命令行启动参数方式**

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

例如：

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

注意：仅对本次hive启动有效

查看参数设置：

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

**（3）SET参数声明方式**

可以在HQL中使用**SET关键字设定参数**

例如：

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

注意：仅对本次hive启动有效。

查看参数设置

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

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

# Hive数据类型

## 数据类型概述

### 基本数据类型

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

对于Hive的String类型相当于数据库的varchar类型，该类型是一个可变的字符串，不过它不能声明其中最多能存储多少个字符，理论上它可以存储2GB的字符数。

### 集合数据类型

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

#### map举例：

假如有如下数据：

001,张三,gg:张二|mm:张四

002,关羽,gg:刘备|dd:张飞

003,吕布,qr:貂蝉|yf:董卓

现在需要建一个hive表来映射数据

如果这样建表：字段扁平化！会造成大量中的有大量的null

而且得对原始数据进行预处理，将各字段展开，没有的字段填空

create table t (id string,name string,gg string, mm string,dd string,qr string,yf)

可以用hive的map类型来映射原始数据中那些亲属关系字段

create table t(id string,name string,relation map<string,string>)

row format delimited fields terminated by ‘,’

collection items terminated by ‘|’

map keys terminated by ‘:’;

加载数据后的结果：

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

| mp.id | mp.name | mp.relation |

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

| 001 | 张三 | {"gg":"张二","mm":"张四"} |

| 002 | 关羽 | {"gg":"刘备","dd":"张飞"} |

| 003 | 吕布 | {"qr":"貂蝉","yf":"董卓"} |

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

#### array数组举例

假如有如下数据：

001,lucy,数学:语文:化学:政治,女

002,rose,数学:语文:政治,女

003,jack,语文:化学,男

现在需要建一个hive表来映射数据

用扁平化字段映射不恰当

可以用数组类型来映射选修科目

create table ar(id string,name string,course array<string>,sex string)

row format delimited fields terminated by ‘,’

collection items terminated by ‘:’;

加载数据后的结果：

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

| ar.id | ar.name | ar.course | ar.sex |

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

| 001 | lucy | ["数学","语文","化学","政治"] | 女 |

| 002 | rose | ["数学","语文","政治"] | 女 |

| 003 | jack | ["语文","化学"] | 男 |

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

#### struct举例

假如有如下数据：

001,张三,18|北京|50

002,关羽,38|运城|85

003,吕布,29|东北|100

现在需要建一个hive表来映射数据

本数据跟上面map示例有点类似，但是区别在于，数据中没有属性key，只有属性值；

如果映射成扁平化字段，可以，但是可能就丢失了这一段数据的整体含义；

可以使用hive的struct结构来映射，不光可以保留整体性，还能为每个属性值添加一个属性名

create table st(id string,name string,info struct<age:int,addr:string,power:int>)

row format delimited fields terminated by ‘,’

collection items terminated by ‘|’

加载数据后的结果：

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

| st.id | st.name | st.info |

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

| 001 | 张三 | {"age":18,"addr":"北京","power":50} |

| 002 | 关羽 | {"age":38,"addr":"运城","power":85} |

| 003 | 吕布 | {"age":29,"addr":"东北","power":100} |

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

## 数据类型详解

### 数字类型

[TINYINT](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types" \l "LanguageManualTypes-tinyint) (1字节整数)

[SMALLINT](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types" \l "LanguageManualTypes-smallint) (2字节整数)

[INT](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types" \l "LanguageManualTypes-int)/INTEGER (4字节整数)

[BIGINT](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types" \l "LanguageManualTypes-bigint) (8字节整数)

FLOAT (4字节浮点数)

DOUBLE (8字节双精度浮点数)

*示例：*

*create table t\_test(a string ,b int,c bigint,d float,e double,f tinyint,g smallint)*

### 时间类型

[TIMESTAMP](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types" \l "LanguageManualTypes-timestamp) (时间戳) (包含年月日时分秒毫秒的一种封装)

[DATE](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types" \l "LanguageManualTypes-date) (日期)（只包含年月日）

示例，假如有以下数据文件：

|  |
| --- |
| 1,zhangsan,1985-06-30  2,lisi,1986-07-10  3,wangwu,1985-08-09 |

那么，就可以建一个表来对数据进行映射

create table t\_customer(id int,name string,birthday date)

row format delimited fields terminated by ',';

然后导入数据

load data local inpath '/root/customer.dat' into table t\_customer;

然后，就可以正确查询

### 字符串类型

[STRING](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types" \l "LanguageManualTypes-string)

[VARCHAR](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types" \l "LanguageManualTypes-varchar) (字符串1-65355长度，超长截断)

[CHAR](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types" \l "LanguageManualTypes-char) (字符串，最大长度255)

### 其他类型

BOOLEAN（布尔类型）：true false

BINARY (二进制)：

### 集合数据类型

array数组类型

arrays: ARRAY<data\_type> )

示例：array类型的应用

假如有如下数据需要用hive的表去映射：

|  |
| --- |
| 战狼2,吴京:吴刚:龙母,2017-08-16  三生三世十里桃花,刘亦菲:痒痒,2017-08-20 |

设想：如果主演信息用一个数组来映射比较方便

建表：

create table t\_movie(moive\_name string,actors array<string>,first\_show date)

row format delimited fields terminated by ','

collection items terminated by ':';

导入数据：

load data local inpath '/root/movie.dat' into table t\_movie;

查询：

select \* from t\_movie;

select moive\_name,actors[0] from t\_movie;

select moive\_name,actors from t\_movie where array\_contains(actors,'吴刚');

select moive\_name,size(actors) from t\_movie;

map类型

maps: MAP<primitive\_type, data\_type>

1. **假如有以下数据：**

|  |
| --- |
| 1,zhangsan,father:xiaoming#mother:xiaohuang#brother:xiaoxu,28  2,lisi,father:mayun#mother:huangyi#brother:guanyu,22  3,wangwu,father:wangjianlin#mother:ruhua#sister:jingtian,29  4,mayun,father:mayongzhen#mother:angelababy,26 |

可以用一个map类型来对上述数据中的家庭成员进行描述

1. **建表语句：**

create table t\_person(id int,name string,family\_members map<string,string>,age int)

row format delimited fields terminated by ','

collection items terminated by '#'

map keys terminated by ':';

1. **查询**

select \* from t\_person;

*## 取map字段的****指定key的值***

select id,name,family\_members['father'] as father

from t\_person;

*## 取map字段的****所有key***

select id,name,map\_keys(family\_members) as relation from t\_person;

*## 取map字段的****所有value***

select id,name,map\_values(family\_members) from t\_person;

select id,name,map\_values(family\_members)[0] from t\_person;

*## 综合：查询有brother的用户信息*

|  |
| --- |
| select id,name,father  from  (select id,name,family\_members['brother'] as brother from t\_person) tmp  where brother is not null;  或者：直接先过滤掉没有brother的  select \* from t\_person where array\_contains(map\_keys(family\_members),'brother'); |

struct类型

structs: STRUCT<col\_name : data\_type, ...>

1. **假如有如下数据：**

|  |
| --- |
| 1,zhangsan,18:male:beijing  2,lisi,28:female:shanghai |

其中的用户信息包含：年龄：整数，性别：字符串，地址：字符串

设想用一个字段来描述整个用户信息，可以采用struct

1. **建表：**

create table t\_person\_struct(id int,name string,info struct<age:int,sex:string,addr:string>)

row format delimited fields terminated by ','

collection items terminated by ':';

1. **查询**

select \* from t\_person\_struct;

select id,name,info.age from t\_person\_struct;

## 类型转化

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

1．隐式类型转换规则如下

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

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

（3）TINYINT、SMALLINT、INT都可以转换为FLOAT。

（4）BOOLEAN类型不可以转换为任何其它的类型。

2．可以使用CAST操作显示进行数据类型转换

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

# DDL 语言

## 创建数据库

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

hive (default)> create database db\_hive;

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

|  |
| --- |
| hive (default)> create database db\_hive;  FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. Database db\_hive already exists  hive (default)> create database if not exists db\_hive; |

3）创建一个数据库，指定数据库在HDFS上的存放位置

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

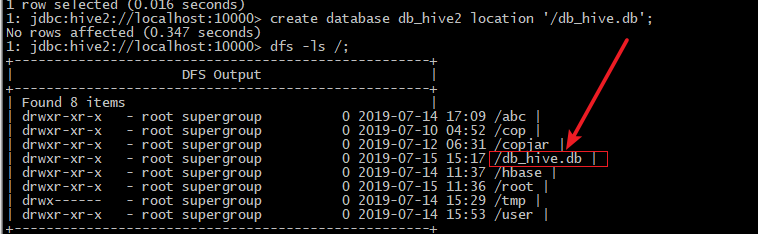


图6-4 数据库存放位置

## 查询数据库

### 显示数据库

1．显示数据库

hive> show databases;

2．通配符显示查询的数据库

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

OK

db\_hive

db\_hive\_1

show databases like '\*lt' ;

| default |

### 查看数据库详情

1．显示数据库信息

hive> desc database db\_hive;

OK

db\_hive hdfs:/doit01:9000/user/hive/warehouse/db\_hive.db

2．显示数据库详细信息，extended

hive> desc database extended db\_hive;

OK

db\_hive hdfs:/doit01:9000/user/hive/warehouse/db\_hive.db

### 切换当前数据库

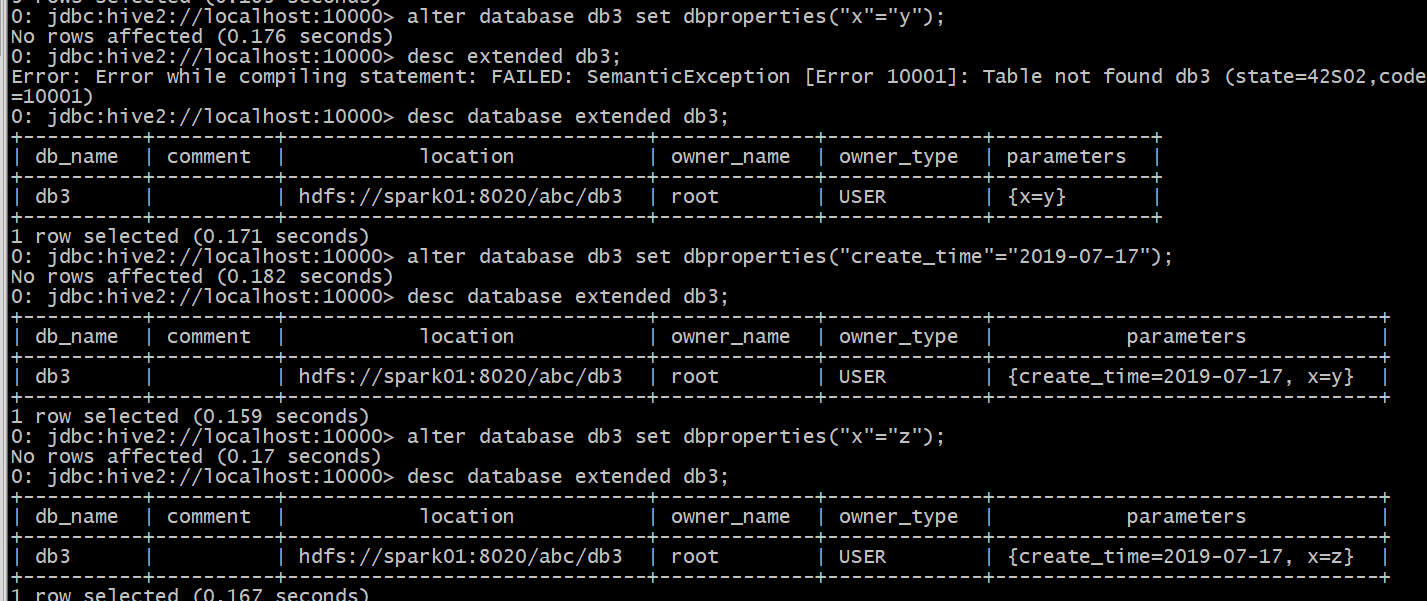
hive (default)> use db\_hive;

## 修改数据库

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

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

在hive中查看修改结果



## 删除数据库

1．删除空数据库

hive>drop database db\_hive2;

2．如果删除的数据库不存在，最好采用 if exists判断数据库是否存在

hive> drop database db\_hive;

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

hive> drop database if exists db\_hive2;

3．如果数据库不为空，可以采用cascade命令，级联删除

hive> drop database db\_hive;

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

hive> drop database db\_hive cascade;

## 创建表

1. 建表语法

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

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

[COMMENT table\_comment]

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

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

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

[ROW FORMAT row\_format] row format delimited fields terminated by “分隔符”

[STORED AS file\_format]

[LOCATION hdfs\_path]

[LIKE]

2．字段解释说明

（1）CREATE TABLE 创建一个指定名字的表。如果相同名字的表已经存在，则抛出异常；用户可以用 IF NOT EXISTS 选项来忽略这个异常。

1. EXTERNAL关键字可以让用户创建一个外部表，在建表的同时指定一个指向实际数据的路径（LOCATION），Hive创建内部表时，会将数据移动到数据仓库指向的路径；若创建外部表，仅记录数据所在的路径，不对数据的位置做任何改变。在删除表的时候，内部表的元数据和数据会被一起删除，而外部表只删除元数据，不删除数据。

（3）COMMENT：为表和列添加注释。

1. PARTITIONED BY创建分区表
2. CLUSTERED BY创建分桶表
3. SORTED BY不常用

（7）ROW FORMAT

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

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

|

SERDE ‘serde\_class\_name’ [WITH SERDEPROPERTIES (property\_name=property\_value, property\_name=property\_value, ...)]

用户在建表的时候可以自定义SerDe或者使用HIVE自带的SerDe。

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

SerDe是Serialize/Deserilize的简称，目的是用于序列化和反序列化。

（8）STORED AS指定存储文件类型

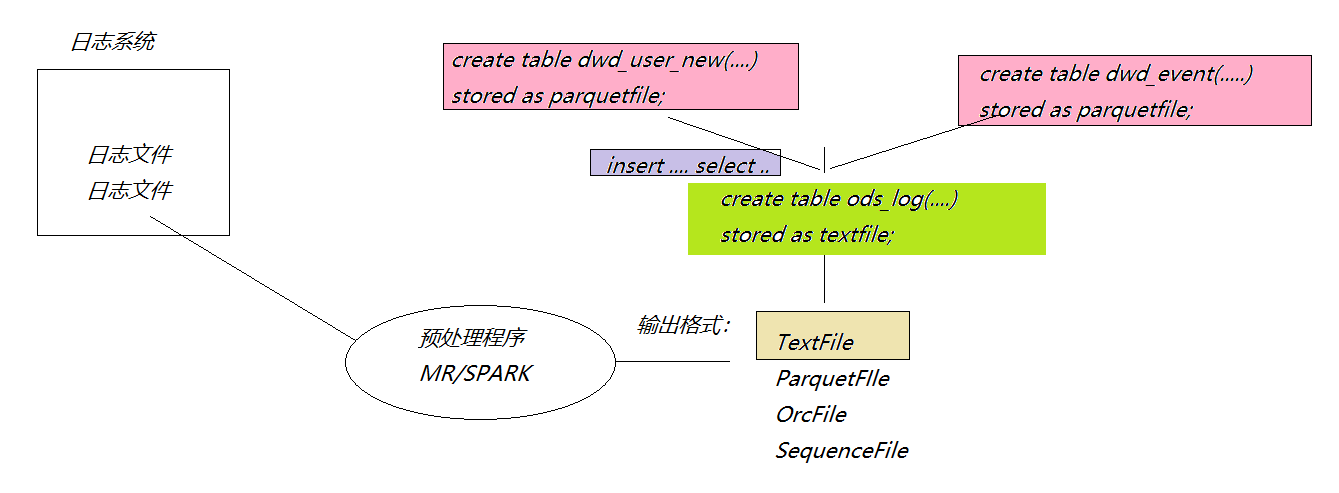
常用的存储文件类型：

SEQUENCEFILE（hadoop\_kv序列文件）、

TEXTFILE（文本）、

RCFILE（列式存储格式文件）、

PARQUETFILE(列式存储文件）



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

1. LOCATION ：指定表在HDFS上的存储位置。

（10）LIKE允许用户复制现有的表结构，但是不复制数据。

### 管理表(内部表)

1．理论

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

2．案例实操

（1）普通创建表

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

（2）根据查询结果创建表（查询的结果会添加到新创建的表中）

|  |
| --- |
| create table if not exists student3 as select id, name from student; |

（3）根据已经存在的表结构创建表

|  |
| --- |
| create table if not exists student4 like student; |

（4）查询表的类型

hive (default)> desc formatted student2;

Table Type: MANAGED\_TABLE

### 外部表

1．理论

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

2．管理表和外部表的使用场景

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

3．案例实操

分别创建部门和员工外部表，并向表中导入数据。

（1）原始数据

（2）建表语句

创建部门表

|  |
| --- |
| create external table if not exists default.dept(  deptno int,  dname string,  loc int  )  row format delimited fields terminated by '\t'; |

创建员工表

|  |
| --- |
| create external table if not exists default.emp(  empno int,  ename string,  job string,  mgr int,  hiredate string,  sal double,  comm double,  deptno int)  row format delimited fields terminated by '\t'; |

（3）查看创建的表

hive (default)> show tables;

OK

tab\_name

dept

emp

（4）向外部表中导入数据

导入数据

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

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

查询结果

hive (default)> select \* from emp;

hive (default)> select \* from dept;

（5）查看表格式化数据

hive (default)> desc formatted dept;

Table Type: EXTERNAL\_TABLE

### 管理表与外部表的互相转换

（1）查询表的类型

hive (default)> desc formatted student2;

Table Type: MANAGED\_TABLE

（2）修改内部表student2为外部表

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

（3）查询表的类型

hive (default)> desc formatted student2;

Table Type: EXTERNAL\_TABLE

（4）修改外部表student2为内部表

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

（5）查询表的类型

hive (default)> desc formatted student2;

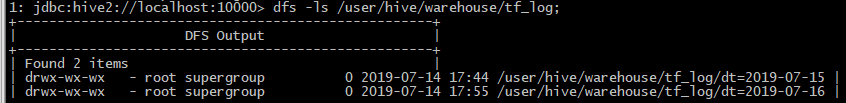
Table Type: MANAGED\_TABLE

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

## 分区表

### 静态分区

分区表实际上就是对应一个HDFS文件系统上的独立的**文件夹**，该文件夹下是该分区所有的数据文件。Hive中的分区就是分目录，把一个大的数据集根据业务需要分割成小的数据集。



在查询时通过WHERE子句中表达式选择查询所需要的指定分区，这样的查询效率会提高很多。

#### 分区表基本操作

1．引入分区表（需要根据日期对日志进行管理）

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

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

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

1. 创建分区表语法

hive (default)> create table dept\_partition(

deptno int, dname string, loc string

)

partitioned by (month string)

row format delimited fields terminated by '\t';

3．加载数据到分区表中

hive (default)> load data local inpath '/opt/module/datas/dept.txt.9' into table default.dept\_partition partition(month='201709');

hive (default)> load data local inpath '/opt/module/datas/dept.txt.8' into table default.dept\_partition partition(month='201708');

hive (default)> load data local inpath '/opt/module/datas/dept.txt' into table default.dept\_partition partition(month='201707’);

4．查询分区表中数据

单分区查询

hive (default)> select \* from dept\_partition where month='201709';

多分区联合查询

hive (default)> select \* from dept\_partition where month='201709'

union

select \* from dept\_partition where month='201708'

union

select \* from dept\_partition where month='201707';

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

10 ACCOUNTING NEW YORK 201707

10 ACCOUNTING NEW YORK 201708

10 ACCOUNTING NEW YORK 201709

20 RESEARCH DALLAS 201707

20 RESEARCH DALLAS 201708

20 RESEARCH DALLAS 201709

30 SALES CHICAGO 201707

30 SALES CHICAGO 201708

30 SALES CHICAGO 201709

40 OPERATIONS BOSTON 201707

40 OPERATIONS BOSTON 201708

40 OPERATIONS BOSTON 201709

5．增加分区

创建单个分区

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

同时创建多个分区

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

6．删除分区

删除单个分区

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

同时删除多个分区

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

7．查看分区表有多少分区

hive> show partitions dept\_partition;

8．查看分区表结构

hive> desc formatted dept\_partition;

# Partition Information

# col\_name data\_type comment

month string

#### 多级分区

定义分区表时，partitioned by后面可以定义多个分区字段；

比如：

create table p2(id int,name string)

partitioned by (**area** string,**dt** string)

row format delimited fields terminated by ',';

它的本质：

在hdfs中的表目录下，会分多层子目录：

示例：

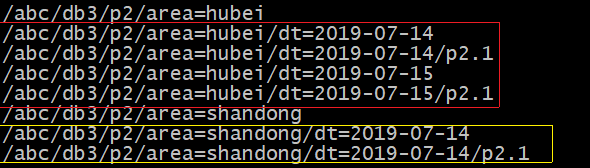
导入数据：

load data local inpath '/root/hivedata/p2.1' into table p2 partition(area='hubei',dt='2019-07-14');

load data local inpath '/root/hivedata/p2.2' into table p2 partition(area='hubei',dt='2019-07-15');

load data local inpath '/root/hivedata/p2.3' into table p2 partition(area='shandong',dt='2019-07-14');

查看存储结构：



#### 分区表注意事项

1．创建二级分区表

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

2．正常的加载数据

（1）加载数据到二级分区表中

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

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

（2）查询分区数据

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

3．把数据导入分区的三种方式

（1）方式一：上传数据后修复

上传数据

hive (default)> dfs -mkdir -p

/user/hive/warehouse/dept\_partition2/month=201709/day=12;

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

/user/hive/warehouse/dept\_partition2/month=201709/day=12;

查询数据（查询不到刚上传的数据）

hive (default)> select \* from dept\_partition2 where month='201709' and day='12';

执行修复命令

hive> msck repair table dept\_partition2;

再次查询数据

hive (default)> select \* from dept\_partition2 where month='201709' and day='12';

（2）方式二：上传数据后添加分区

上传数据

hive (default)> dfs -mkdir -p

/user/hive/warehouse/dept\_partition2/month=201709/day=11;

hive (default)> dfs -put /opt/module/datas/dept.txt /user/hive/warehouse/dept\_partition2/month=201709/day=11;

执行添加分区

hive (default)> alter table dept\_partition2 add partition(month='201709',

day='11');

查询数据

hive (default)> select \* from dept\_partition2 where month='201709' and day='11';

（3）方式三：创建文件夹后load数据到分区

创建目录

hive (default)> dfs -mkdir -p

/user/hive/warehouse/dept\_partition2/month=201709/day=10;

上传数据

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

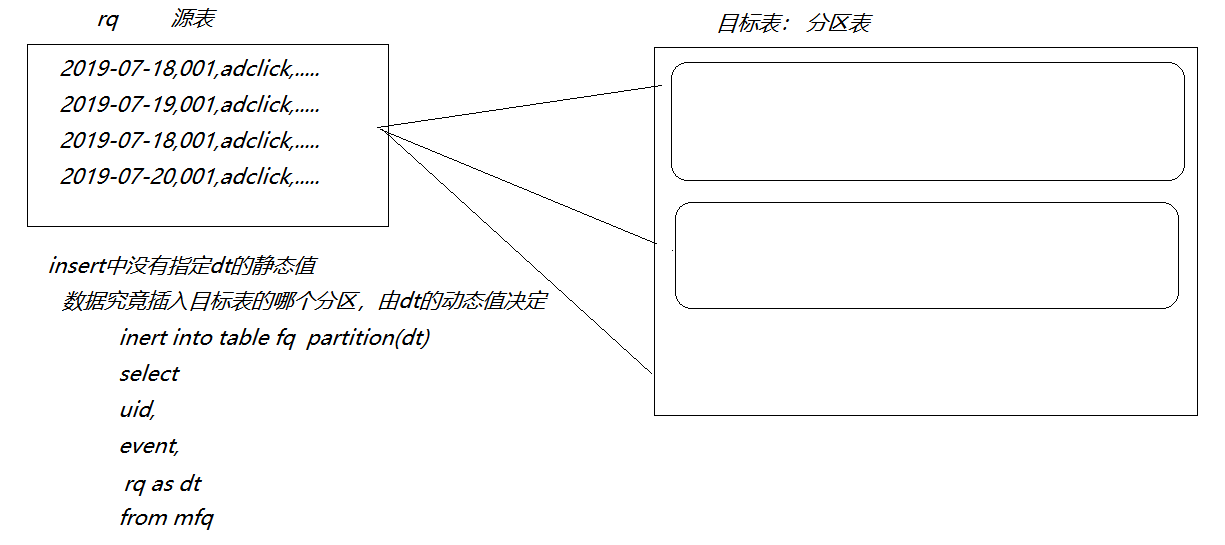
dept\_partition2 partition(month='201709',day='10');

查询数据

hive (default)> select \* from dept\_partition2 where month='201709' and day='10';

### 动态分区

按照某个字段的不同值自动将数据加载到不同分区中



1） 创建一个作为数据源的测试表

create table demo(

id int ,

birthday string ,

cost int

)

row format delimited fields terminated by '\t' ;

2）加载数据

load data local inpath '/data/demo' into table demo ;

3）数据

1 2010-11-12 120

2 2010-11-12 121

3 2010-11-12 122

4 2010-11-12 124

5 2010-11-13 122

6 2010-11-13 120

7 2010-11-14 123

8 2010-11-13 1202

9 2010-11-12 1202

10 2010-11-12 1201

4）创建测试目标：分区表

create table demo2(

id int ,

cost int

) partitioned by(x string)

row format delimited fields terminated by '\t' ;

5）设置参数

set hive.exec.dynamic.partition=true //使用动态分区

set hive.exec.dynamic.partition.mode=nonstrick;//无限制模式，如果模式是strict，则必须有一个静态分区且放在最前面

set hive.exec.max.dynamic.partitions.pernode=10000;//每个节点生成动态分区的最大个数

set hive.exec.max.dynamic.partitions=100000;//生成动态分区的最大个数

set hive.exec.max.created.files=150000;//一个任务最多可以创建的文件数目

~~set dfs.datanode.max.xcievers=8192;//限定一次最多打开的文件数~~

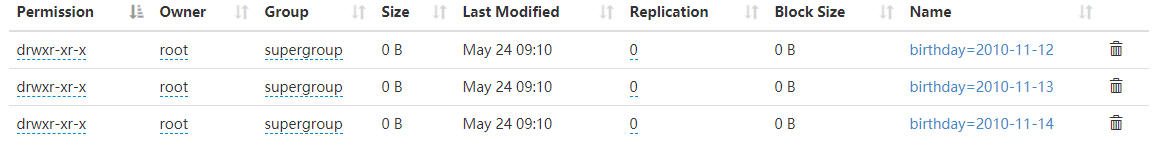
set hive.merge.mapfiles=true; // map端的结果进行合并

set mapreduce.reduce.tasks =20000; //设置reduce task个数 增加reduce阶段的并行度

6）加载数据

insert into table demo2 partition(x) --此处的分区变量x应该跟demo2中的分区变量名一致

select id , cost ,birthday from demo; --select中的最后一个表达式birthday，会作为分区变量x的动态值，注意顺序



总结：

1. 动态分区的语法，见上
2. 动态分区的底层机制：

mapreduce中的多路输出mutipleOutputs(根据条件判断，将结果写入不同目录不同文件）

1. 动态分区的意义：

不知道分区字段究竟有多少种类型时，使用静态分区不好实现；

用动态分区，则可以自动完成数据的划分！

## 修改表定义（元数据）

### 重命名表

1．语法

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

2．实操案例

hive (default)> alter table x rename to y;

### 增加、修改和删除表分区

详见4.6.1分区表基本操作。

alter table x add partition(dt=’09’)

alter table x drop partition(dt=’09’)

### 增加/修改/替换列定义信息

1．语法

更新列

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

增加和替换列

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

注：ADD是代表新增一字段，字段位置在所有列后面(partition列前)，REPLACE则是表示替换表中所有字段。

2．实操案例

（1）查询表结构

hive> desc dept\_partition;

（2）添加列

hive (default)> alter table x add columns(deptdesc string); // 该字段添加到字段顺序的最后

（3）查询表结构

hive> desc dept\_partition;

（4）更新列

hive (default)> alter table dept\_partition change column deptdesc desc int;

（5）查询表结构

hive> desc dept\_partition;

（6）替换列

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

string, loc string);

（7）查询表结构

hive> desc dept\_partition;

## 删除表

hive (default)> drop table dept\_partition;

## SerDe组件

* **简介**

1、SerDe是Serialize/Deserilize的简称，目的是用于序列化和反序列化。

2、用户在建表时可以用自定义的SerDe或使用Hive自带的SerDe，SerDe能为表**切分、解析列**，且对列指定相应的数据。

* **示例如下：**

数据文件内容

id=123,name=steven

id=55,name=ray

期望输出：

123 steven

55 ray

1. 创建表, 用正则表达式的形式指定格式

create table test1(id int, name string)

row format serde 'org.apache.hadoop.hive.serde2.RegexSerDe'

with serdeproperties ("input.regex" = "id=(.\*),name=(.\*)")

stored as textfile;

2. 加载数据

load data local inpath '/root/hivedata/test1.dat’ overwrite into table test1;

3. 查询结果

更多SerDe实现，可参考官方文档：

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

https://cwiki.apache.org/confluence/display/Hive/DeveloperGuide#DeveloperGuide-HiveSerDe

LazySimpleSerDe ：https://issues.apache.org/jira/browse/HIVE-292?jql=text ~ "LazySimpleSerDe"

ColumnarSerDe ： https://issues.apache.org/jira/browse/HIVE-756?jql=text ~ "ColumnarSerDe"

RegexSerDe ：https://issues.apache.org/jira/browse/HIVE-6336?jql=text ~ "RegexSerDe"

HBaseSerDe ：https://issues.apache.org/jira/browse/HIVE-6677?jql=text ~ "HBaseSerDe"

AvroSerDe ： https://issues.apache.org/jira/issues/?jql=text ~ "AvroSerDe"

ParquetHiveSerDe：https://issues.apache.org/jira/browse/HIVE-9333?jql=text ~ "ParquetHiveSerde"

OpenCSVSerDe ：https://issues.apache.org/jira/browse/HIVE-7777?jql=text ~ "OpenCSVSerDe"

JSONSerDe ： https://issues.apache.org/jira/browse/HIVE-6166?jql=text

# 数据导入导出操作

## 数据导入

### 使用load向表中装载数据

1．语法

hive> load data [local] inpath '/opt/module/datas/student.txt' overwrite | into table student [partition (partcol1=val1,…)];

（1）load data:表示加载数据

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

（3）inpath:表示加载数据的路径

（4）overwrite:表示覆盖表中已有数据，否则表示追加

（5）into table:表示加载到哪张表

（6）student:表示具体的表

（7）partition:表示上传到指定分区

2．实操案例

（1）创建一张表

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

（2）加载本地文件到hive

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

（3）加载HDFS文件到hive中

上传文件到HDFS

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

加载HDFS上数据

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

（4）加载数据覆盖表中已有的数据

上传文件到HDFS

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

加载数据覆盖表中已有的数据

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

### 使用insert...values 向表中插入数据

hive (default)> insert into table student partition(month='201709') values(1,'wangwu');

### 使用insert...select向表中插入数据

1．创建一张分区表

hive (default)> create table hang3(id int, name string) partitioned by (month string) row format delimited fields terminated by '\t';

2．基本模式插入（根据单张表查询结果）

hive (default)> insert overwrite table student partition(month='201708')

select id, name from student where month='201709';

3．多插入模式（根据多张表查询结果）

hive (default)> from student

insert overwrite table student partition(month='201709')

select id, name where month='201709'

insert overwrite table student partition(month='201708')

select id, name where month='201708';

### 用create as创建表并加载数据

根据查询结果创建表（查询的结果会添加到新创建的表中）

create table if not exists student3

as

select id, name from student;

### 创建表时通过指定location加载数据路径

1．创建表，并指定在hdfs上的位置

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

id int, name string

)

row format delimited fields terminated by '\t'

location '/user/hive/warehouse/student5';

2．上传数据到hdfs上

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

/user/hive/warehouse/student5;

3．查询数据

hive (default)> select \* from student5;

### import数据到指定Hive表中

注意：先用export导出后，再将数据导入。

hive (default)> import table student2 partition(month='201709') from

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

## 数据导出

### insert导出

1．将查询的结果导出到本地

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

select \* from student;

2．将查询的结果格式化导出到本地

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

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

3．将查询的结果导出到HDFS上(没有local)

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

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'

select \* from student;

### ~~hadoop命令导出到本地~~

hive (default)> dfs -get /user/hive/warehouse/student/month=201709/000000\_0

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

### ~~hive shell 命令导出~~

基本语法：（hive -f/-e 执行语句或者脚本 > file）

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

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

### Export导出到HDFS上

(defahiveult)> export table default.student to

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

### Sqoop导出到其他关系型数据库

后续课程专门讲。

## 清除表中数据（Truncate）

注意：Truncate只能删除管理表，不能删除外部表中数据

hive (default)> truncate table student;

# SQL快速掌握

## sql的运算模型 -- 逐行运算模型

逐行运算：select后的运算表达式，是对每一行独立运算

表：一个数据的集合（集合中每一行就是一条数据：记录）

select：

对一条数据的运算逻辑

-- 常量 :"ok" ,10 ,8.9

-- 变量 : id , name , age,

-- 运算符表达式： age+10, id+8 , id>8 , id>2 and id<9

-- 函数表达式： upper(name) , greatest(s1,s2,s3)

-- 复合表达式： greatest(s1,s2,s3)>2 , greatest(s1,s2,s3)+10 , lower(substr(upper(name),0,3))

from ：选择一个运算的数据集

join：准备数据集（它可以将多个数据集拼成一个数据集，拼的时候可以带条件）

但是，join的拼接有多种方式：

t1 join t2 --> 笛卡尔积,结果的总行数的t1的行数\*t2的行数；

t1 join t2 on t1.id=t2.id -->内连接，满足拼接条件的才拼接..

t1 left join t2 on t1.id=t2.id --> 左（外）连接 left outer join ;左表所有行都保留，连接不上的右表字段为null

t1 right join t2 on t1.id=t2.id -->右（外）连接 right outer join;右表的所有行都保留，连接不上的左表字段为null

t1 full join t2 on t1.id=t2.id -->全（外）连接 full outer join;左、右表的行都保留，拼不上的字段为null

-- hive不支持不等值join

-- hive中有一种特别的join

left semi join -->左半连接，是sql中in子句的一个变种实现！

hive1.x中不支持in，现在的新版本hive支持in子句

where：逐行过滤将要运算的数据集,where执行在select之前

-- where id>2

-- where (id+10)>2

-- where upper(name) = 'ZHANGSAN'

-- where id in (select id from t\_x)

group by：

函数：类似java代码中的方法，接收变量，返回结果，也就是一个表达式

having：

紧跟group by之后，是对分组后的数据进行按组过滤，将不满足条件表达式的组去除；

## sql运算模型--分组聚合运算

有如下数据：

1,zs1,M,18,18800

2,zs2,M,28,22800

3,zs3,F,38,21800

4,zs4,F,19,25000

5,zs5,M,26,24000

6,zs6,F,30,19000

需求：

男生的薪资总和

女生的薪资总和

age,salary

1,zs1,hebei,M,18,18800

2,zs2,hebei,M,28,22800 ==> 65600

5,zs5,hebei,M,26,24000

3,zs3,hebei,F,38,21800

4,zs4,hebei,F,19,25000 ==> 65800

6,zs6,hebei,F,30,19000

select "ok",sex,sum(salary)

from t1

group by sex;

-- 分组运算模型中，select后面的表达式只能有如下情形：

1. 常量

2. 分组key

3. 聚合函数

/\* 分组key可以有多个，分组key越多，分出来的组也会越多 \*/

2,zs2,河南,M,28,22800

5,zs5,河南,M,26,24000

1,zs1,河北,M,18,18800

3,zs3,河北,F,38,21800

4,zs4,山东,F,19,25000

6,zs6,山东,F,30,19000

求：每一个省份中的 男生薪资总和及 女生薪资总和

select province,sex,sum(salary)

from t1

group by province,sex

/\* 统计每个省各性别薪资总和，但薪资低于2万的不参与统计 \*/

// where 过滤一定要在group by之前

select

from t1

where salary>20000

group by province,sex

/\* 统计每个省各性别薪资总和，结果中不需要总和低于4万的 \*/

-- 子查询--嵌套查询

select

from

(

select province,sex,sum(salary) as amt

from t1

group by province,sex

) o

where o.amt > 40000

-- 不用子查询，使用分组过滤 having

select

province,sex,sum(salary)

from t1

group by province,sex

having sum(salary)>40000

2,zs2,河南,M,28,22800 2,zs2,河南,M,28,22800

5,zs5,河南,M,26,24000 ==> sum(salary)>40000 ==> true 5,zs5,河南,M,26,24000

1,zs1,河北,M,18,18800 ==> sum(salary)>40000 ==> false -------------------

3,zs3,河北,F,38,21800 ==> sum(salary)>40000 ==> false -------------------

4,zs4,山东,F,19,25000 4,zs4,山东,F,19,25000

6,zs6,山东,F,30,19000 ==> sum(salary)>40000 ==> true 6,zs6,山东,F,30,19000

/\* \*/

drop table if exists x1;

create table x1(id int,jb int,tc int,jt int)

row format delimited fields terminated by ',';

load data local inpath '/root/hivedata/x1.dat' into table x1;

x1

id,基本工资,提成,津贴

1,2000,3000,1500

2,5000,500,1000

3,1500,1000,3000

4,3000,6000,8000

5,1500,2000,1800

x2

create table x2(id int,gw string)

row format delimited fields terminated by ',';

load data local inpath '/root/hivedata/x2.dat' into table x2;

id,岗位

1,销售

2,技术

3,行政

4,销售

5,行政

x3

create table x3(id int,name string,sex string,age int)

row format delimited fields terminated by ',';

load data local inpath '/root/hivedata/x3.dat' into table x3;

id,name,sex,age

1,zs,M,28

2,ww,F,36

3,zl,F,48

4,pp,M,44

5,wb,M,32

/\* 求出公司中每个员工的三类收入中最高的那种收入 \*/

select id,greatest(`基本工资`,`提成`,`津贴`)

from t1;

/\* 求出公司中每个岗位的薪资总和 \*/

select t2.`岗位`,sum(`基本工资`+`提成`+`津贴`)

from

t1 join t2 on t1.id=t2.id

group by t2.`岗位`

;

/\* 求出公司中每个岗位不同性别员工薪资总和 \*/

select t2.`岗位`,t3.sex,sum(`基本工资`+`提成`+`津贴`)

from

t1 join t2 on t1.id=t2.id

join t3 on t2.id=t3.id

group by t2.`岗位`,t3.sex

;

/\* 求出公司中不同性别、不同年龄阶段（20-30,31-40,41-50）的员工薪资总和 \*/

select sex,jieduan,sum(base+tc+jt) as amt

from

(

select

x1.id as x1id,

x1.jb as base,

x1.tc as tc,

x1.jt as jt,

x3.id as x3id,

x3.name as name,

x3.sex as sex,

x3.age as age,

case

when x3.age>=20 and x3.age<=30 then '20-30'

when x3.age>=31 and x3.age<=40 then '31-40'

else '41-50'

end as jieduan

from

x1 join x3 on x1.id=x3.id

) o

group by sex,jieduan

;

/\* 求出每种性别的员工的最高收入类别的薪资总和 \*/

select x3.sex,sum(greatest(jb,tc,jt))

from x1 join x3 on x1.id=x3.id

group by x3.sex

;

/\* 求每种性别中，总薪资 最高 的那个人的 "id,name,性别，age,总薪资" \*/

select o3.id,o3.name,o3.sex,o3.age,o2.max\_s

from

(

select

sex,

max(salary) as max\_s

from

(

select

x1.id as id,

x3.name as name,

x3.sex as sex,

x3.age as age,

x1.jb+x1.tc+x1.jt as salary

from

x1 join x3 on x1.id=x3.id

) o1

group by sex

) O2

JOIN

(

select

x1.id as id,

x3.name as name,

x3.sex as sex,

x3.age as age,

x1.jb+x1.tc+x1.jt as salary

from

x1 join x3 on x1.id=x3.id

) o3

ON o2.max\_s = o3.salary and o2.sex=o3.sex

;

## sql运算模型--开窗运算（窗口分析运算模型）

-- 窗口分析模型不像分组运算模型一样，体现在select groupby的结构上，而是体现在窗口分析函数上

create table y1(id int,name string,sex string,age int,salary int)

row format delimited fields terminated by ',';

load data local inpath '/root/hivedata/y1.dat' into table y1;

1,zs,M,16,2000

2,ls,F,18,3000

3,ww,M,19,2060

4,zl,F,28,1800

5,tq,F,38,2800

6,wb,M,22,2500

7,jj,M,16,2000

8,ss,F,18,3000

/\* 请求出每种性别中，薪资最高的两个人的信息 \*/

-- 如果上表的数据能产生如下的中间结果，就好求了：

6,wb,M,22,2500,1

3,ww,M,19,2060,2

1,zs,M,16,2000,3

7,jj,M,16,2000,4

2,ls,F,18,3000,1

8,ss,F,18,3000,2

5,tq,F,38,2800,3

4,zl,F,28,1800,4

-- 可以用窗口分析函数：row\_number() over() 来实现

select

id,name,sex,age,salary,

row\_number() over(partition by sex order by salary desc) as rn

from y1;

--然后在此基础上，得出每种性别中薪资最高的两个人的信息：

select id,name,sex,age,salary

from

(

select

id,name,sex,age,salary,

row\_number() over(partition by sex order by salary desc) as rn

from y1

) o

where rn<=2

;

/\* \*/

-- 加强练习 ：打地鼠游戏记录求连续命中次数

创建测试表：

create table ds(uid string,seq int,m int) row format delimited fields terminated by ',';

load data local inpath '/root/hivedata/ds.dat' into table ds;

数据如下：

uid,seq,m

u01,1,1

u01,2,0

u01,3,1

u01,4,1

u01,5,0

u01,6,1

u02,1,1

u02,2,1

u02,4,1

u02,5,1

u02,8,1

u02,9,1

u03,1,1

u03,2,1

u03,3,1

u03,4,1

u03,5,1

u03,6,0

-- 步骤1： 过滤掉没有命中的记录

select uid,seq,m

from ds

where m=1;

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

| uid | seq | m |

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

| u01 | 1 | 1 |

| u01 | 3 | 1 |

| u01 | 4 | 1 |

| u01 | 6 | 1 |

| u02 | 1 | 1 |

| u02 | 2 | 1 |

| u02 | 4 | 1 |

| u02 | 5 | 1 |

| u02 | 8 | 1 |

| u02 | 9 | 1 |

| u03 | 1 | 1 |

| u03 | 2 | 1 |

| u03 | 3 | 1 |

| u03 | 4 | 1 |

| u03 | 5 | 1 |

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

-- 步骤2,： 为用户的命中记录，打上连续的行号

select uid,seq,m,

row\_number() over(partition by uid order by seq) as rn

from

(

select uid,seq,m

from ds

where m=1) o1

;

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

| uid | seq | m | rn |

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

| u01 | 1 | 1 | 1 |

| u01 | 3 | 1 | 2 |

| u01 | 4 | 1 | 3 |

| u01 | 6 | 1 | 4 |

| u02 | 1 | 1 | 1 |

| u02 | 2 | 1 | 2 |

| u02 | 4 | 1 | 3 |

| u02 | 5 | 1 | 4 |

| u02 | 8 | 1 | 5 |

| u02 | 9 | 1 | 6 |

| u03 | 1 | 1 | 1 |

| u03 | 2 | 1 | 2 |

| u03 | 3 | 1 | 3 |

| u03 | 4 | 1 | 4 |

| u03 | 5 | 1 | 5 |

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

-- 步骤3：用次序号-rn行号得到一个差值

select uid,seq,m,rn,seq-rn as diff

from

(

select uid,seq,m,

row\_number() over(partition by uid order by seq) as rn

from

(

select uid,seq,m

from ds

where m=1) o1

) o2

;

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

| uid | seq | m | rn || diff |

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

| u01 | 1 | 1 | 1 || 0 |

| u01 | 3 | 1 | 2 || 1 |

| u01 | 4 | 1 | 3 || 1 |

| u01 | 6 | 1 | 4 || 2 |

| u02 | 1 | 1 | 1 || 0 |

| u02 | 2 | 1 | 2 || 0 |

| u02 | 4 | 1 | 3 || 1 |

| u02 | 5 | 1 | 4 || 1 |

| u02 | 8 | 1 | 5 || 3 |

| u02 | 9 | 1 | 6 || 3 |

| u03 | 1 | 1 | 1 || 0 |

| u03 | 2 | 1 | 2 || 0 |

| u03 | 3 | 1 | 3 || 0 |

| u03 | 4 | 1 | 4 || 0 |

| u03 | 5 | 1 | 5 || 0 |

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

-- 步骤4：分组求同一个人的各类差值的条数

select uid,diff,count(1)

from

(

select uid,seq,m,rn,seq-rn as diff

from

(

select uid,seq,m,

row\_number() over(partition by uid order by seq) as rn

from

(

select uid,seq,m

from ds

where m=1) o1

) o2

) o3

group by uid,diff

-- having count(1)>=3

;

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

| uid | diff | \_c2 |

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

| u01 | 0 | 1 |

| u01 | 1 | 2 |

| u01 | 2 | 1 |

| u02 | 0 | 2 |

| u02 | 1 | 2 |

| u02 | 3 | 2 |

| u03 | 0 | 5 |

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

-- 步骤5：对满足连续命中>=3的uid去重

SELECT DISTINCT uid

FROM (

SELECT uid

,diff

,count(1) AS cnt

FROM (

SELECT uid

,seq

,m

,rn

,seq - rn AS diff

FROM (

SELECT uid

,seq

,m

,row\_number() OVER (

PARTITION BY uid ORDER BY seq

) AS rn

FROM (

SELECT uid

,seq

,m

FROM ds

WHERE m = 1

) o1

) o2

) o3

GROUP BY uid

,diff

-- having count(1)>=3

) o4

WHERE cnt >= 3;

-- 同类拓展题：求： 每家店铺连续销售天数的最大记录

a,2019-02-01,300

a,2019-02-02,500

a,2019-02-03,550

a,2019-02-05,400

a,2019-02-06,500

b,2019-02-01,300

b,2019-02-02,500

b,2019-02-03,550

b,2019-02-04,400

b,2019-02-05,500

# HQL查询语法详解

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

查询语句语法：

[WITH CommonTableExpression (, CommonTableExpression)\*]    (Note: Only available

starting with Hive 0.13.0)

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

  FROM table\_reference

  [WHERE where\_condition]

  [GROUP BY col\_list]

  [ORDER BY col\_list]

  [CLUSTER BY col\_list

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

  ]

 [LIMIT number]

## with..as临时表语法

示例：

with o as (

select

id,m,sum(sale) as amt

from t\_sale

group by id,m

)

select \* from o where o.amt>80

## 基本查询（Select…From）

### 全表和特定列查询

1．全表查询

hive (default)> select \* from emp;

2．选择特定列查询

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

注意：

（1）SQL 语言大小写不敏感。

（2）SQL 可以写在一行或者多行

（3）关键字不能被缩写也不能分行

（4）各子句一般要分行写。

（5）使用缩进提高语句的可读性。

### 列别名

1．重命名一个列

2．便于计算

3．紧跟列名，也可以在列名和别名之间加入关键字‘AS’

4．案例实操

查询名称和部门

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

### Limit语句

典型的查询会返回多行数据。LIMIT子句用于限制返回的行数。

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

### distinct

去重

典型错误写法： select name,distinct id from t1;

正确写法： select distinct id,name from t1; // 对id+name组合去重

## Where语句

1．使用WHERE子句，将不满足条件的行过滤掉

2．WHERE子句紧随FROM子句

3．案例实操

查询出薪水大于1000的所有员工

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

## 各类运算符

### 算术运算符

表6-3

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

案例实操

查询出所有员工的薪水后加1显示。

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

### 比较运算符（Between/In/ Is Null）

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

|  |  |  |
| --- | --- | --- |
| 操作符 | 支持的数据类型 | 描述 |
| A=B | 基本数据类型 | 如果A等于B则返回TRUE，反之返回FALSE |
| A<=>B | 基本数据类型 | 如果A和B都为NULL，则返回TRUE，其他的和等号（=）操作符的结果一致，如果任一为NULL则结果为NULL |
| A<>B, A!=B | 基本数据类型 | A或者B为NULL则返回NULL；如果A不等于B，则返回TRUE，反之返回FALSE |
| A<B | 基本数据类型 | A或者B为NULL，则返回NULL；如果A小于B，则返回TRUE，反之返回FALSE |
| A<=B | 基本数据类型 | A或者B为NULL，则返回NULL；如果A小于等于B，则返回TRUE，反之返回FALSE |
| A>B | 基本数据类型 | A或者B为NULL，则返回NULL；如果A大于B，则返回TRUE，反之返回FALSE |
| A>=B | 基本数据类型 | A或者B为NULL，则返回NULL；如果A大于等于B，则返回TRUE，反之返回FALSE |
| A [NOT] BETWEEN B AND C | 基本数据类型 | 如果A，B或者C任一为NULL，则结果为NULL。如果A的值大于等于B而且小于或等于C，则结果为TRUE，反之为FALSE。如果使用NOT关键字则可达到相反的效果。 |
| A IS NULL | 所有数据类型 | 如果A等于NULL，则返回TRUE，反之返回FALSE |
| A IS NOT NULL | 所有数据类型 | 如果A不等于NULL，则返回TRUE，反之返回FALSE |
| IN(数值1, 数值2) | 所有数据类型 | 使用 IN运算显示列表中的值 |
| A [NOT] LIKE B | STRING 类型 | B是一个SQL下的简单正则表达式，如果A与其匹配的话，则返回TRUE；反之返回FALSE。B的表达式说明如下：‘x%’表示A必须以字母‘x’开头，‘%x’表示A必须以字母’x’结尾，而‘%x%’表示A包含有字母’x’,可以位于开头，结尾或者字符串中间。如果使用NOT关键字则可达到相反的效果。 |
| A RLIKE B, A REGEXP B | STRING 类型 | B是一个正则表达式，如果A与其匹配，则返回TRUE；反之返回FALSE。匹配使用的是JDK中的正则表达式接口实现的，因为正则也依据其中的规则。例如，正则表达式必须和整个字符串A相匹配，而不是只需与其子符串匹配。 |

2）案例实操

（1）查询出薪水等于5000的所有员工

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

（2）查询工资在500到1000的员工信息

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

（3）查询comm为空的所有员工信息

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

（4）查询工资是1500或5000的员工信息

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

### Like和RLike

1）使用LIKE运算选择类似的值

2）选择条件可以包含字符或数字:

% 代表零个或多个字符(任意个字符)。

\_ 代表一个字符。

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

4）案例实操

（1）查找以2开头薪水的员工信息

hive (default)> select \* from emp where sal LIKE '2%';

（2）查找第二个数值为2的薪水的员工信息

hive (default)> select \* from emp where sal LIKE '\_2%';

（3）查找薪水中含有2的员工信息

hive (default)> select \* from emp where sal RLIKE '[2]';

select \*

from tu

where url rlike 'http://www.doitedu.cn/job[?,/]';

### 逻辑组合运算符（And/Or/Not）

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

**拼接的join的ON后面不能使用or连接**

**hive中不支持 ON的不等连接**

案例实操

（1）查询薪水大于1000，部门是30

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

（2）查询薪水大于1000，或者部门是30

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

（3）查询除了20部门和30部门以外的员工信息

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

## 分组

### Group By语句

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

案例实操：

（1）计算emp表每个部门的平均工资

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

（2）计算emp每个部门中每个岗位的最高薪水

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

t.deptno, t.job;

### Having语句

1．having与where不同点

（1）where针对表中的列发挥作用，查询数据；having针对查询结果中的列发挥作用，筛选数据。

（2）where后面不能写分组函数，而having后面可以使用分组函数。

（3）having只用于group by分组统计语句。

2．案例实操

（1）求每个部门的平均薪水大于2000的部门

求每个部门的平均工资

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

求每个部门的平均薪水大于2000的部门

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

avg\_sal > 2000;

## Join语句

### 等值Join

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

案例实操

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

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

on e.deptno = d.deptno;

### 表的别名

1．好处

（1）使用别名可以简化查询。

（2）使用表名前缀可以提高执行效率。

2．案例实操

合并员工表和部门表

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

= d.deptno;

### 内连接

内连接：只有进行连接的两个表中都存在与连接条件相匹配的数据才会被保留下来。

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

= d.deptno;

### 左外连接

左外连接：JOIN操作符左边表中符合WHERE子句的所有记录将会被返回。

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

= d.deptno;

### 右外连接

右外连接：JOIN操作符右边表中符合WHERE子句的所有记录将会被返回。

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

= d.deptno;

### 全外连接

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

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

= d.deptno;

### 多表连接

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

数据准备



1．创建位置表

|  |
| --- |
| create table if not exists location(  loc int,  loc\_name string  )  row format delimited fields terminated by '\t'; |

2．导入数据

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

3．多表连接查询

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

FROM emp e

JOIN dept d

ON d.deptno = e.deptno

JOIN location l

ON d.loc = l.loc;

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

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

### 笛卡尔积

1．笛卡尔集会在下面条件下产生

**~~（1）省略连接条件~~**

（2）连接条件无效

（3）所有表中的所有行互相连接

2．案例实操

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

### 连接谓词中不建议or

因为用 join on *t1.id=t2.id OR t1.addr=t2.addr*

这种条件的表达，不是一个确定的唯一的连接条件，不好表达成mapreduce

它只能将两个表进行笛卡尔积连接，然后再对连接后的数据进行过滤

效率极其低下；

if you know what you are doing, you can ：

set hive.strict.checks.cartesian.product=false;

## 排序

### 全局排序（Order By）

select \* from t\_sale order by sale desc,m asc,id asc;

Order By：**全局排序**，强制只有一个Reducer

1．使用 ORDER BY 子句排序

ASC（ascend）: 升序（默认）

DESC（descend）: 降序

2．ORDER BY 子句在SELECT语句的结尾

3．案例实操

（1）查询员工信息按工资升序排列

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

（2）查询员工信息按工资降序排列

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

### 按照别名排序

按照员工薪水的2倍排序

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

### 多个列排序

按照部门和工资升序排序

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

### （Sort By）

Sort By：每个task内部进行排序，对全局结果集来说不是排序。

1．设置reduce个数

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

2．查看设置reduce个数

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

3．根据部门编号降序查看员工信息

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

4．将查询结果导入到文件中（按照部门编号降序排序）

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

select \* from emp sort by deptno desc;

### 分桶排序查询（Distribute By + Sort by）

Distribute By：类似MR中partition，进行分区，可以结合sort by使用。

注意，Hive要求DISTRIBUTE BY语句要写在SORT BY语句之前。

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

案例实操：

（1）先按照部门编号分区，再按照员工编号降序排序。

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

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

### 分桶排序Cluster By

当distribute by和sorts by字段相同时，可以使用cluster by代替。

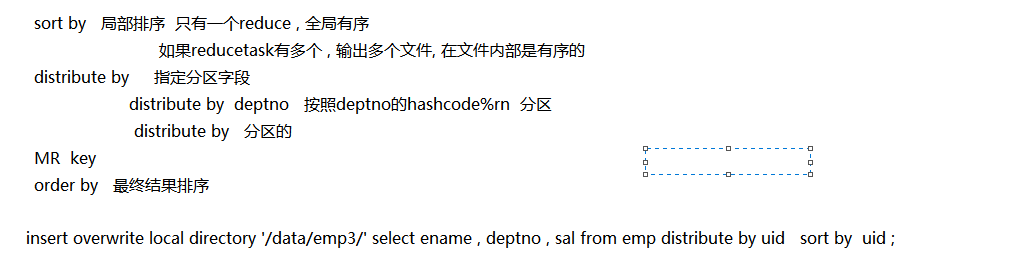
cluster by除了具有distribute by的功能外还兼具sort by的功能。但是排序只能是升序排序，不能指定排序规则为ASC或者DESC。

1）以下两种写法等价

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

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

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



## 分桶及抽样查询

### 分桶表的定义

对Hive表分桶可以将表中数据按分桶键的哈希值散列到多个文件中，这些小文件称为桶。

表分区是用不同的子**文件夹**管理不同的数据；

表分桶用不同的**文件**管理不同的数据。

分桶是将数据集分解成更容易管理的若干部分的另一个技术。

create table ft\_x(id int,name string,province string,city string,age int)

**clustered by** (province) **sorted by** (age) into **30** buckets

row format delimited fields terminated by ‘,’;

如何向分桶表中加载数据：

方式1： 直接导入文件（不建议）

直接导入的文件，可能不符合分桶表的分桶规范！

如果非要导入，先设置参数：

hive> set hive.strict.checks.bucketing=false;

然后即可导入；

方式2： 通过一个中转表，用insert.....select.....来导入数据

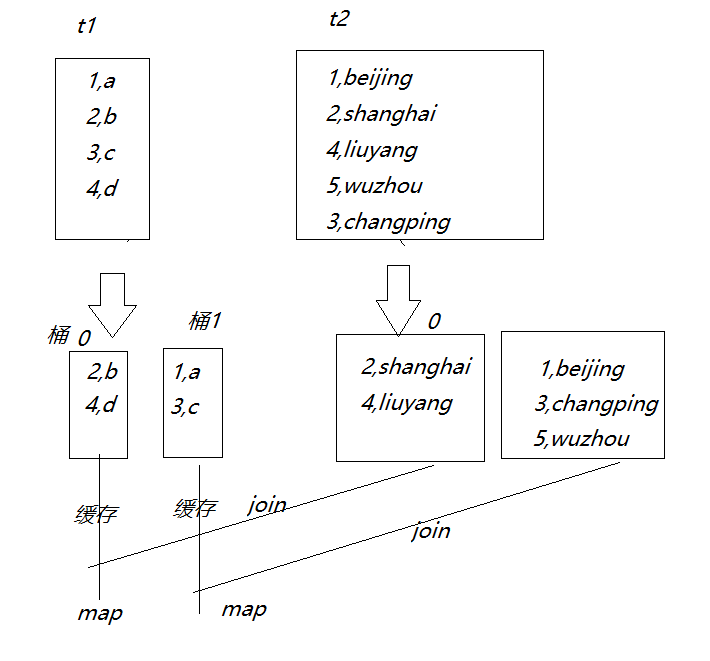
检查参数： set mapreduce.job.reduces= -1(或目标分桶表的桶数);  
hive> insert into ft\_x

> select \* from pt;

### 分桶表的意义

把表或分区划分成bucket有两个理由

join更快，join相同列划分的桶的表，可以使用map-side join；而且能提高计算的并行度；更加高效。



1. 取样sampling更高效、方便。没有分桶的话需要扫描整个数据集。

|  |
| --- |
| hive (default)> select \* from buck\_demo **tablesample**(bucket 1 out of 4 on id); |

### 分桶抽样查询

对于非常大的数据集，有时用户需要使用的是一个具有代表性的查询结果而不是全部结果。Hive可以通过对表进行抽样来满足这个需求。

查询表stu\_buck中的数据。

|  |
| --- |
| hive (default)> select \* from buck\_demo **tablesample**(bucket 1 out of 4 on id); |

注：tablesample是抽样语句，语法：TABLESAMPLE(BUCKET x OUT OF y) 。

y必须是table总bucket数的倍数或者因子。

hive根据**y的大小，决定抽样的比例**。例如，table总共分了4桶

当y=2时，抽取(4/2=)2个bucket的数据

当y=8时，抽取(4/8=)1/2个bucket的数据。

x表示从哪个bucket开始抽取，如果需要取多个分区，以后的分区号为当前分区号加上y。

例如，table总bucket数为4，tablesample(bucket 1 out of 2)，表示总共抽取（4/2=）2个bucket的数据，抽取第1(x)个和第3(x+y)个bucket的数据。

注意：x的值必须小于等于y的值，否则

FAILED: SemanticException [Error 10061]: Numerator should not be bigger than denominator in sample clause for table stu\_buck

# 函数

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF?searchId=4811FCH61

## 内置函数帮助命令

1．查看系统自带的函数

hive> show functions;

2．显示自带的函数的用法

hive> desc function upper;

3．详细显示自带的函数的用法

hive> desc function extended upper;

## 常用单行运算函数

### 类型转换函数

select cast("5" as int) ~~;~~

select cast("2017-08-03" as date) ;

select cast(current\_timestamp as date);

*示例：*

|  |  |  |
| --- | --- | --- |
| *1* | *1995-05-05 13:30:59* | *1200.3* |
| *2* | *1994-04-05 13:30:59* | *2200* |
| *3* | *1996-06-01 12:20:30* | *80000.5* |

*create table t\_fun(id string,birthday string,salary string)*

*row format delimited fields terminated by ',';*

*select id,cast(birthday as date) as bir,cast(salary as float) from t\_fun;*

### 数学运算函数

select round(5.4); ## 5 四舍五入

select round(5.1345,3) ; ##5.135

select ceil(5.4) ; // select ceiling(5.4) from dual; ## 6 向上取整

select floor(5.4); ## 5 向下取整

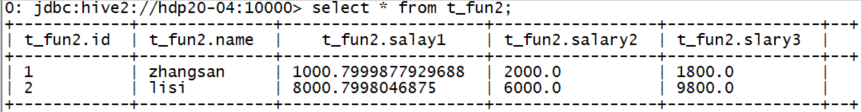
select abs(-5.4) ; ## 5.4 绝对值

select greatest(3,5,6) ; ## 6

select least(3,5,6) ; ##求多个输入参数中的最小值

示例：

有表如下：



select greatest(cast(s1 as double),cast(s2 as double),cast(s3 as double)) from t\_fun2;

结果：

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

| \_c0 |

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

| 2000.0 |

| 9800.0 |

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

select max(age) from t\_person; 聚合函数

select min(age) from t\_person; 聚合函数

### 时间函数

select current\_timestamp; ## 获取当前的时间戳(详细时间信息)

select current\_date; ## 获取当前的日期

## unix时间戳转字符串格式——**from\_unixtime**

**from\_unixtime**(bigint unixtime[, string format])

示例：select from\_unixtime(unix\_timestamp());

select from\_unixtime(unix\_timestamp(),"yyyy/MM/dd HH:mm:ss");

## 字符串格式转unix时间戳——**unix\_timestamp**

## 如果不带参数，取当前时间的秒数时间戳--(距离格林威治时间1970-1-1 0:0:0秒的差距)

select **unix\_timestamp()**;

**unix\_timestamp**(string date, string pattern)

示例： select unix\_timestamp("2017-08-10 17:50:30");

select unix\_timestamp("2017-08-10 17:50:30","yyyy-MM-dd HH:mm:ss");

## 将字符串转成日期date

select **to\_date**("2017-09-17 16:58:32");

**datediff 求日期差**

select datediff(‘2019-07-16’,’2019-07-14’)

**date\_add 日期往后推N天**

select date\_add(‘2019-07-16’,3)

**date\_sub 日期往前推N天**

select date\_sub(‘2019-07-16’,3)

**to\_date 把字符串日期变date类型的日期**

select to\_date(‘2019-07-21 19:30:20’) => 2019-07-21

**next\_day 求下一个星期几是几号**

select next\_day(‘2019-07-18’,’monday’) => 2019-07-22

**weekofyear 求指定日期在一年中是第几周**

select weekofyear(‘2019-07-18’) => 29

### 字符串函数

select base64(binary('13800001111')); ##字符串变成base64编码

substr(string str, int start) ## 截取子串

substring(string str, int start)

*示例：select substr("abcdefg",2) from dual;*

**substr(string, int start, int len)**

substring(string, int start, int len)

*示例：select substr("abcdefg",2,3) from dual;*

**concat**(string A, string B...) ## 拼接字符串

**concat\_ws**(string SEP, string A, string B...)

*示例：select concat("ab","xy") from dual; ## abxy*

*select concat\_ws(".","192","168","33","44") from dual; ## 192.168.33.44*

**length**(string A)

*示例：select length("192.168.33.44") from dual; ## 13*

**split(string str, string pat)**

示例：~~select split("192.168.33.44",".") ;~~ 错误的，因为.号是正则语法中的特定字符

select split("192.168.33.44","\\.") from dual;

**upper**(string str) ##转大写

**lower**(string str) ##转小写

lpad(string A,int len,String pad) ## 在A的左边填充pad，填充到长度len

rpad(string A,int len,String pad) ## 在A的右边填充pad，填充到长度len

ltrim(string A)

rtrim(String A)

trim(String A)

**get\_json\_object ## json解析**

// 有一个表中，x字段为json字符串

{"movie":"1193","rate":"5","timeStamp":"978300760","uid":"1"}

利用get\_json\_object可以方便地对x字段进行解析展开

select get\_json\_object(json,"$.movie"),

get\_json\_object(json,"$.rate"),

get\_json\_object(json,"$.timeStamp"),

get\_json\_object(json,"$.uid")

from tjson;

### 条件分支函数CASE

语法：

CASE [ expression ]

WHEN condition1 THEN result1

WHEN condition2 THEN result2

...

WHEN conditionn THEN resultn

ELSE result

END

示例：

select id,name,

case

when age<28 then 'youngth'

when age>27 and age<40 then 'zhongnian'

else 'old'

end

from t\_user;

select

id,m,day,sale,

case sale

when 20 then '小额'

when 30 then '中额'

else '大额'

end as e

from t\_sale

;

### 条件分支函数IF

select id,if(age>25,'working','worked') from t\_user;

select moive\_name,if(array\_contains(actors,'吴刚'),'好电影',’烂片儿’)

from t\_movie;

### 集合函数

**array\_contains**(Array<T>, value) 返回boolean值

示例：

select moive\_name,array\_contains(actors,'吴刚') from t\_movie;

select array\_contains(array('a','b','c'),'c') from dual;

**sort\_array**(Array<T>) 返回排序后的数组

*示例：*

select sort\_array(array('c','b','a')) from dual;

select 'haha',sort\_array(array('c','b','a')) as xx from (select 0) tmp;

**size**(Array<T>) 返回一个集合的长度，int值

*示例：*

select moive\_name,size(actors) as actor\_number from t\_movie;

**size**(Map<K.V>) 返回一个imap的元素个数，int值

size(array<T>) 返回一个数组的长度,int值

**map\_keys**(Map<K.V>) 返回一个map字段的所有key，结果类型为：数组

**map\_values**(Map<K.V>) 返回一个map字段的所有value，结果类型为：数组

### 空字段赋值

1. 函数说明

NVL：给值为NULL的数据赋值，它的格式是NVL( string1, replace\_with)。它的功能是如果string1为NULL，则NVL函数返回replace\_with的值，否则返回string1的值，如果两个参数都为NULL ，则返回NULL。

2.示例：

数据准备：采用员工表

1）查询：如果员工的comm为NULL，则用-1代替

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

OK

\_c0

20.0

300.0

500.0

-1.0

1400.0

-1.0

-1.0

-1.0

-1.0

0.0

-1.0

-1.0

-1.0

-1.0

2）查询：如果员工的comm为NULL，则用领导id代替

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

OK

\_c0

20.0

300.0

500.0

7839.0

1400.0

7839.0

7839.0

7566.0

NULL

0.0

7788.0

7698.0

7566.0

## 窗口分析函数

### 基本语法说明

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

CURRENT ROW：当前行 current row

n PRECEDING：往前n行数据 n preceding

n FOLLOWING：往后n行数据 n following

UNBOUNDED：起点，UNBOUNDED PRECEDING 表示从前面的起点， UNBOUNDED FOLLOWING表示到后面的终点 unbound preceding unbound following

LAG(col,n)：往前第n行数据 lag 参数一 字段 n

LEAD(col,n)：往后第n行数据 lead

### sum over案例演示

1．数据准备：name，orderdate，cost

jack,2017-01-01,10

tony,2017-01-02,15

jack,2017-02-03,23

tony,2017-01-04,29

jack,2017-01-05,46

jack,2017-04-06,42

tony,2017-01-07,50

jack,2017-01-08,55

mart,2017-04-08,62

mart,2017-04-09,68

neil,2017-05-10,12

mart,2017-04-11,75

neil,2017-06-12,80

mart,2017-04-13,94

2．需求

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

4．创建本地business.txt，导入数据

[doit@hadoop102 datas]$ vi business.txt

5．创建hive表并导入数据

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

6．按需求查询数据

（1）查询在2017年4月份购买过的顾客及总人数

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

（2）查询顾客的购买明细及月购买总额

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

（3）上述的场景,要将cost按照日期进行累加

|  |
| --- |
| select name,orderdate,cost,  sum(cost) over() as sample1,--所有行相加  sum(cost) over(partition by name) as sample2,--按name分组，组内数据相加  sum(cost) over(partition by name order by orderdate) as sample3,--按name分组，组内数据累加  sum(cost) over(partition by name order by orderdate rows between UNBOUNDED PRECEDING and current row ) as sample4 ,--和sample3一样,由起点到当前行的聚合  sum(cost) over(partition by name order by orderdate rows between 1 PRECEDING and current row) as sample5, --当前行和前面一行做聚合  sum(cost) over(partition by name order by orderdate rows between 1 PRECEDING AND 1 FOLLOWING ) as sample6,--当前行和前边一行及后面一行  sum(cost) over(partition by name order by orderdate rows between current row and UNBOUNDED FOLLOWING ) as sample7 --当前行及后面所有行  from business; |

查看顾客上次的购买时间

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

查询前20%时间的订单信息

|  |
| --- |
| select  \*  from  (  select  \* ,  ntile(5) over(order by cday) n  from  orders  ) t  where t.n = 1; |

### rank案例

1．函数说明

RANK() 排序相同时会重复，总数不会变

DENSE\_RANK() 排序相同时会重复，总数会减少

ROW\_NUMBER() 会根据顺序计算

2．数据准备

表6-7 数据准备

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

3．需求

计算每门学科成绩排名。

4．创建本地score.txt，导入数据

[doit@hadoop102 datas]$ vi score.txt

5．创建hive表并导入数据

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

6．按需求查询数据

|  |
| --- |
| select name,  subject,  score,  rank() over(partition by subject order by score desc) rp,  dense\_rank() over(partition by subject order by score desc) drp,  row\_number() over(partition by subject order by score desc) rmp  from score;  name subject score rp drp rmp  孙悟空 数学 95 1 1 1  宋宋 数学 86 2 2 2  婷婷 数学 85 3 3 3  娜娜 数学 56 4 4 4  宋宋 英语 84 1 1 1  娜娜 英语 84 1 1 2  婷婷 英语 78 3 2 3  孙悟空 英语 68 4 3 4  娜娜 语文 94 1 1 1  孙悟空 语文 87 2 2 2  婷婷 语文 65 3 3 3  宋宋 语文 64 4 4 4 |

### lead over

### lag over

### sum over

## 分组聚合函数

sum(字段)

avg

max

min

count

1．求总行数（count）

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

2．求工资的最大值（max）

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

3．求工资的最小值（min）

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

4．求工资的总和（sum）

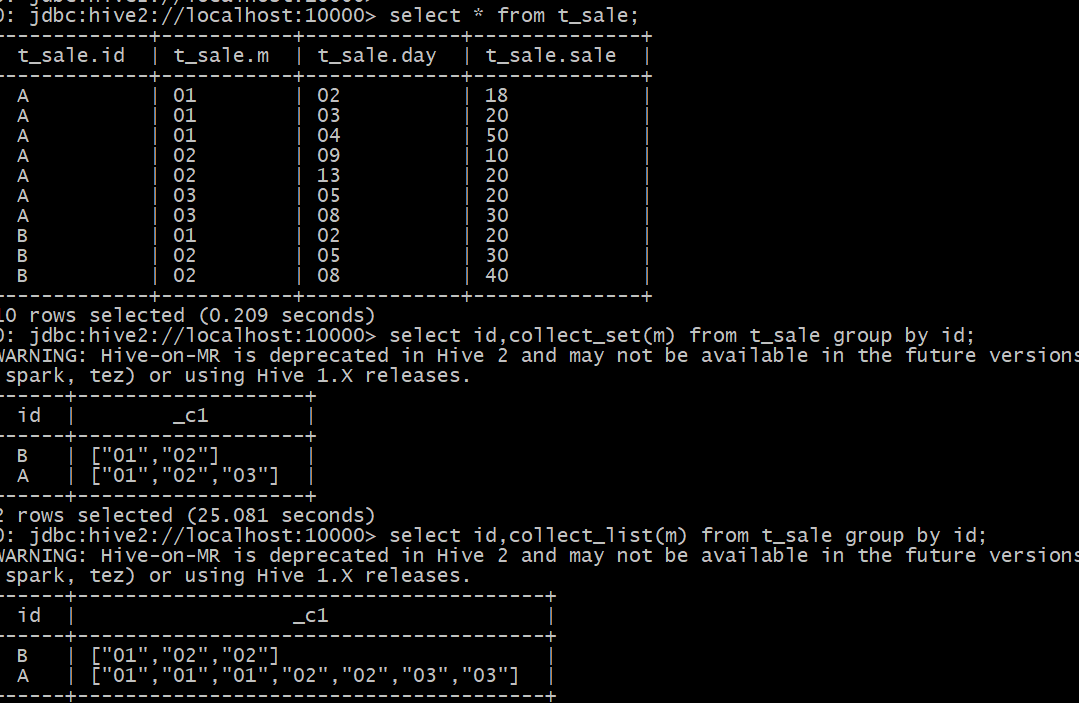
hive (default)> select sum(sal) sum\_sal from emp;

5．求工资的平均值（avg）

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

collect\_set

collect\_list



## 表生成函数

### 行转列函数：explode()

假如有以下数据：

|  |
| --- |
| 1,zhangsan,化学:物理:数学:语文  2,lisi,化学:数学:生物:生理:卫生  3,wangwu,化学:语文:英语:体育:生物 |

映射成一张表：

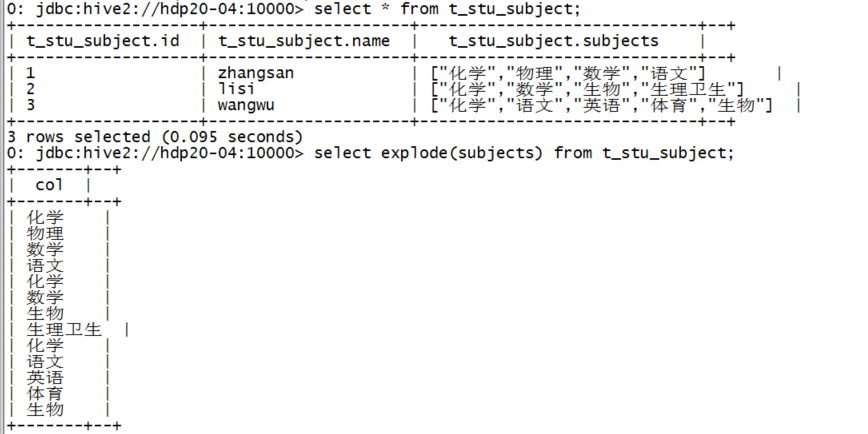
create table t\_stu\_subject(id int,name string,subjects array<string>)

row format delimited fields terminated by ','

collection items terminated by ':';

load data local inpath ‘/root/sb.dat’ into table t\_stu\_subject;

使用explode()对数组字段“炸裂”



然后，我们利用这个explode的结果，来求去重的课程：

|  |
| --- |
| select distinct tmp.sub  from  (select explode(subjects) as sub from t\_stu\_subject) tmp; |

### 表生成函数lateral view

select id,name,tmp.sub

from t\_stu\_subject lateral view explode(subjects) tmp as sub;



*理解： lateral view 相当于两个表在join*

*左表：是原表*

*右表：是explode(某个集合字段)之后产生的表*

*而且：这个join只在同一行的数据间进行*

*那样，可以方便做更多的查询：*

*比如，查询选修了生物课的同学*

*select a.id,a.name,a.sub from*

(select id,name,tmp.sub as sub from t\_stu\_subject lateral view explode(subjects) tmp as sub) a

where sub='生物';

另一个完整的示例：

|  |
| --- |
| 0: jdbc:hive2://localhost:10000> select \* from ar;  +--------+----------+------------------------+---------+  | ar.id | ar.name | ar.course | ar.sex |  +--------+----------+------------------------+---------+  | 001 | lucy | ["数学","语文","化学","政治"] | 女 |  | 002 | rose | ["数学","语文","政治"] | 女 |  | 003 | jack | ["语文","化学"] | 男 |  +--------+----------+------------------------+---------+  3 rows selected (0.169 seconds)  0: jdbc:hive2://localhost:10000> select explode(course) from ar;  +------+  | col |  +------+  | 数学 |  | 语文 |  | 化学 |  | 政治 |  | 数学 |  | 语文 |  | 政治 |  | 语文 |  | 化学 |  +------+  9 rows selected (0.136 seconds)  0: jdbc:hive2://localhost:10000> select id,name,explode(course) from ar;  Error: Error while compiling statement: FAILED: SemanticException [Error 10081]: UDTF's are not supported outside the SELECT clause, nor nested in expressions (state=42000,code=10081)  0: jdbc:hive2://localhost:10000> select id,name,course,sex  . . . . . . . . . . . . . . . .> from  . . . . . . . . . . . . . . . .> ar  . . . . . . . . . . . . . . . .> lateral view  . . . . . . . . . . . . . . . .> explode(course) tmp as c  . . . . . . . . . . . . . . . .> ;  +------+-------+------------------------+------+  | id | name | course | sex |  +------+-------+------------------------+------+  | 001 | lucy | ["数学","语文","化学","政治"] | 女 |  | 001 | lucy | ["数学","语文","化学","政治"] | 女 |  | 001 | lucy | ["数学","语文","化学","政治"] | 女 |  | 001 | lucy | ["数学","语文","化学","政治"] | 女 |  | 002 | rose | ["数学","语文","政治"] | 女 |  | 002 | rose | ["数学","语文","政治"] | 女 |  | 002 | rose | ["数学","语文","政治"] | 女 |  | 003 | jack | ["语文","化学"] | 男 |  | 003 | jack | ["语文","化学"] | 男 |  +------+-------+------------------------+------+  9 rows selected (0.131 seconds)  0: jdbc:hive2://localhost:10000> select id,name,course,sex,tmp.c  . . . . . . . . . . . . . . . .> from  . . . . . . . . . . . . . . . .> ar  . . . . . . . . . . . . . . . .> lateral view  . . . . . . . . . . . . . . . .> explode(course) tmp as c  . . . . . . . . . . . . . . . .> ;  +------+-------+------------------------+------+--------+  | id | name | course | sex | tmp.c |  +------+-------+------------------------+------+--------+  | 001 | lucy | ["数学","语文","化学","政治"] | 女 | 数学 |  | 001 | lucy | ["数学","语文","化学","政治"] | 女 | 语文 |  | 001 | lucy | ["数学","语文","化学","政治"] | 女 | 化学 |  | 001 | lucy | ["数学","语文","化学","政治"] | 女 | 政治 |  | 002 | rose | ["数学","语文","政治"] | 女 | 数学 |  | 002 | rose | ["数学","语文","政治"] | 女 | 语文 |  | 002 | rose | ["数学","语文","政治"] | 女 | 政治 |  | 003 | jack | ["语文","化学"] | 男 | 语文 |  | 003 | jack | ["语文","化学"] | 男 | 化学 |  +------+-------+------------------------+------+--------+  9 rows selected (0.095 seconds)  0: jdbc:hive2://localhost:10000> select id,name,sex,tmp.c  . . . . . . . . . . . . . . . .> from  . . . . . . . . . . . . . . . .> ar  . . . . . . . . . . . . . . . .> lateral view  . . . . . . . . . . . . . . . .> explode(course) tmp as c  . . . . . . . . . . . . . . . .> ;  +------+-------+------+--------+  | id | name | sex | tmp.c |  +------+-------+------+--------+  | 001 | lucy | 女 | 数学 |  | 001 | lucy | 女 | 语文 |  | 001 | lucy | 女 | 化学 |  | 001 | lucy | 女 | 政治 |  | 002 | rose | 女 | 数学 |  | 002 | rose | 女 | 语文 |  | 002 | rose | 女 | 政治 |  | 003 | jack | 男 | 语文 |  | 003 | jack | 男 | 化学 |  +------+-------+------+--------+  9 rows selected (0.085 seconds)  0: jdbc:hive2://localhost:10000> |

### json解析函数：表生成函数

需求：有如下json格式的电影评分数据：

|  |
| --- |
| {"movie":"1193","rate":"5","timeStamp":"978300760","uid":"1"}  {"movie":"661","rate":"3","timeStamp":"978302109","uid":"1"}  {"movie":"914","rate":"3","timeStamp":"978301968","uid":"1"}  {"movie":"3408","rate":"4","timeStamp":"978300275","uid":"1"}  {"movie":"2355","rate":"5","timeStamp":"978824291","uid":"1"}  {"movie":"1197","rate":"3","timeStamp":"978302268","uid":"1"} |

需要做各种统计分析。

发现，直接对json做sql查询不方便，需要将json数据解析成普通的结构化数据表。可以采用hive中内置的json\_tuple()函数

实现步骤：

1. 创建一个原始表用来对应原始的json数据

create table t\_json(json string);

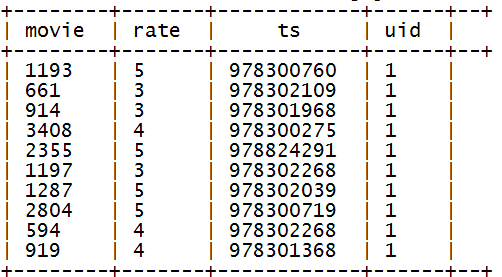
load data local inpath ‘/root/rating.json’ into table t\_json;

1. 利用json\_tuple进行json数据解析

测试，示例：

select json\_tuple(json,'movie','rate','timeStamp','uid') as(movie,rate,ts,uid) from t\_json limit 10;

产生结果：



真正解析整张json表，将解析结果数据插入一张新表

create table t\_movie\_rate

as

select json\_tuple(json,'movie','rate','timeStamp','uid') as(movie,rate,ts,uid) from t\_json;

-- TODO 练习--

1/ 统计每部电影的平均得分

create table t\_rate\_m\_avg as select movie,avg(rate) from t\_rate group by movie;

2/ 统计数据中一共有多少部电影

select count(distinct movie) from t\_rate;

3/ 统计每部电影的被评分次数（热门度）

create table t\_rate\_m\_cnt as select movie,count(1) as cnt from t\_rate group by movie order by cnt desc;

4/ 统计每个人的所有电影评分的平均值

create table t\_rate\_u\_avg as select uid,avg(rate) from t\_rate group by uid;

5/ 查询出每个人评分最高的20部电影信息

利用json\_tuple从原始json数据表中，etl出一个详细信息表：

|  |
| --- |
| create table t\_rate  as  select  uid,  movie,  rate,  year(from\_unixtime(cast(ts as bigint))) as year,  month(from\_unixtime(cast(ts as bigint))) as month,  day(from\_unixtime(cast(ts as bigint))) as day,  hour(from\_unixtime(cast(ts as bigint))) as hour,  minute(from\_unixtime(cast(ts as bigint))) as minute,  from\_unixtime(cast(ts as bigint)) as ts  from  (select  json\_tuple(rateinfo,'movie','rate','timeStamp','uid') as(movie,rate,ts,uid)  from t\_json) tmp  ; |

## 自定义函数

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

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

3）根据用户自定义函数类别分为以下三种：

（1）UDF（User-Defined-Function）

一进一出

（2）UDAF（User-Defined Aggregation Function）

聚集函数，多进一出

类似于：count/max/min

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

一进多出

如lateral view explore() split()

4）官方文档地址

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

5）编程步骤：

（1）继承org.apache.hadoop.hive.ql.UDF

（2）需要实现evaluate函数；evaluate函数支持重载；

（3）在hive的命令行窗口创建函数

a）添加jar

add jar linux\_jar\_path

b）创建function，

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

（4）在hive的命令行窗口删除函数

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

6）注意事项

（1）UDF必须要有返回类型，可以返回null，但是返回类型不能为void；

## 自定义UDF函数示例

1．创建一个Maven工程Hive

2．导入依赖

|  |
| --- |
| <dependencies>  <!-- https://mvnrepository.com/artifact/org.apache.hive/hive-exec -->  <dependency>  <groupId>org.apache.hive</groupId>  <artifactId>hive-exec</artifactId>  <version>2.3.5</version>  </dependency>  </dependencies> |

1. 创建一个类

|  |
| --- |
| package com.doit.hive;  import org.apache.hadoop.hive.ql.exec.UDF;  public class Lower extends UDF {  public String evaluate (final String s) {    if (s == null) {  return null;  }  return s.toLowerCase();  }  } |

4．打成jar包上传到服务器/opt/module/jars/udf.jar

5．将jar包添加到hive的classpath

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

6．创建临时函数与开发好的java class关联

hive (default)> create temporary function mf as "com.\_51doit.func.MyFunction";

7．即可在hql中使用自定义的函数strip

hive (default)> select ename, mylower(ename) lowername from emp;

永久函数的使用:  
　　1. 把自定义函数的jar上传到hdfs中.  
　　hdfs dfs -put lower.jar 'hdfs:///path/to/hive\_func';  
　　2. 创建永久函数  
　　hive> create function xxoo\_lower as 'com.\_51doit.func.MyFunction' using jar 'hdfs:///path/to/hive\_func/lower.jar'  
　　3. 验证  
　　hive> select xxoo\_lower("Hello World");  
　　hive> show functions;  
　　永久函数的删除也容易:  
　　hive> drop function xxoo\_lower;

# 压缩和存储

## ~~Hadoop源码编译支持Snappy压缩~~

### ~~资源准备~~

1．CentOS联网

配置CentOS能连接外网。Linux虚拟机ping [www.baidu.com](http://www.baidu.com) 是畅通的

注意：采用root角色编译，减少文件夹权限出现问题

2．jar包准备(hadoop源码、JDK8 、maven、protobuf)

（1）hadoop-2.7.2-src.tar.gz

（2）jdk-8u144-linux-x64.tar.gz

（3）snappy-1.1.3.tar.gz

（4）apache-maven-3.0.5-bin.tar.gz

（5）protobuf-2.5.0.tar.gz

### ~~jar包安装~~

注意：所有操作必须在root用户下完成

1. JDK解压、配置环境变量JAVA\_HOME和PATH，验证[java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "_blank) -version(如下都需要验证是否配置成功)

|  |
| --- |
| [root@doit01 software] # tar -zxf jdk-8u144-linux-x64.tar.gz -C /opt/module/  [root@doit01 software]# vi /etc/profile  #JAVA\_HOME  export JAVA\_HOME=/opt/module/jdk1.8.0\_144  export PATH=$PATH:$JAVA\_HOME/bin  [root@doit01 software]#source /etc/profile |

验证命令：java -version

2．Maven解压、配置  MAVEN\_HOME和PATH

|  |
| --- |
| [root@doit01 software]# tar -zxvf apache-maven-3.0.5-bin.tar.gz -C /opt/module/  [root@doit01 apache-maven-3.0.5]# vi /etc/profile  #MAVEN\_HOME  export MAVEN\_HOME=/opt/module/apache-maven-3.0.5  export PATH=$PATH:$MAVEN\_HOME/bin  [root@doit01 software]#source /etc/profile |

验证命令：mvn -version

### ~~编译源码~~

1．准备编译环境

[root@doit01 software]# yum install svn

[root@doit01 software]# yum install autoconf automake libtool cmake

[root@doit01 software]# yum install ncurses-devel

[root@doit01 software]# yum install openssl-devel

[root@doit01 software]# yum install gcc\*

2．编译安装snappy

[root@doit01 software]# tar -zxvf snappy-1.1.3.tar.gz -C /opt/module/

[root@doit01 module]# cd snappy-1.1.3/

[root@doit01 snappy-1.1.3]# ./configure

[root@doit01 snappy-1.1.3]# make

[root@doit01 snappy-1.1.3]# make install

# 查看snappy库文件

[root@doit01 snappy-1.1.3]# ls -lh /usr/local/lib |grep snappy

3．编译安装protobuf

[root@doit01 software]# tar -zxvf protobuf-2.5.0.tar.gz -C /opt/module/

[root@doit01 module]# cd protobuf-2.5.0/

[root@doit01 protobuf-2.5.0]# ./configure

[root@doit01 protobuf-2.5.0]# make

[root@doit01 protobuf-2.5.0]# make install

# 查看protobuf版本以测试是否安装成功  
[root@doit01 protobuf-2.5.0]# protoc --version

4．编译hadoop native

[root@doit01 software]# tar -zxvf hadoop-2.7.2-src.tar.gz

[root@doit01 software]# cd hadoop-2.7.2-src/

[root@doit01 software]# mvn clean package -DskipTests -Pdist,native -Dtar -Dsnappy.lib=/usr/local/lib -Dbundle.snappy

执行成功后，/opt/software/hadoop-2.7.2-src/hadoop-dist/target/[hadoop](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "_blank)-2.7.2.tar.gz即为新生成的支持snappy压缩的二进制安装包。

## Hadoop压缩配置

### MR支持的压缩编码

表6-8

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

为了支持多种压缩/解压缩算法，Hadoop引入了编码/解码器，如下表所示：

表6-9

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

压缩性能的比较：

表6-10

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

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

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

### 压缩参数配置

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

表6-11

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

## 开启Map输出阶段压缩

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

**案例实操：**

1．开启hive中间传输数据压缩功能

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

2．开启mapreduce中map输出压缩功能

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

3．设置mapreduce中map输出数据的压缩方式

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

org.apache.hadoop.io.compress.SnappyCodec;

4．执行查询语句

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

## 开启Reduce输出阶段压缩

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

**案例实操：**

1．开启hive最终输出数据压缩功能

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

2．开启mapreduce最终输出数据压缩

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

3．设置mapreduce最终数据输出压缩方式

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

org.apache.hadoop.io.compress.GzipCodec;

4．设置mapreduce最终数据输出压缩为块压缩

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

5．测试一下输出结果是否是压缩文件

hive (default)> insert overwrite local directory

'/data' select \* from shop;

## 文件存储格式

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

### 列式存储和行式存储

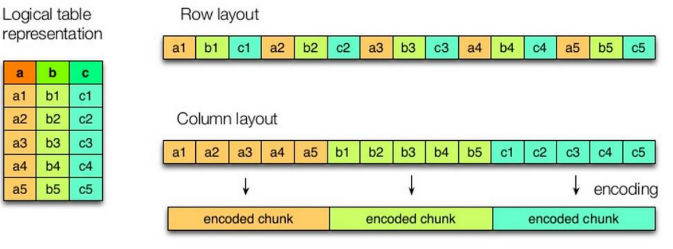


图6-10 列式存储和行式存储

如图6-10所示左边为逻辑表，右边第一个为行式存储，第二个为列式存储。

1．行存储的特点

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

2．列存储的特点

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

TEXTFILE和SEQUENCEFILE的存储格式都是基于行存储的；

ORC和PARQUET是基于列式存储的。

### TextFile格式

默认格式，数据不做压缩，磁盘开销大，数据解析开销大。可结合Gzip、Bzip2使用，但使用Gzip这种方式，hive不会对数据进行切分，从而无法对数据进行并行操作。

### Orc格式

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

如图6-11所示可以看到每个Orc文件由1个或多个stripe组成，每个stripe250MB大小，这个Stripe实际相当于RowGroup概念，不过大小由4MB->250MB，这样应该能提升顺序读的吞吐率。每个Stripe里有三部分组成，分别是Index Data，Row Data，Stripe Footer：

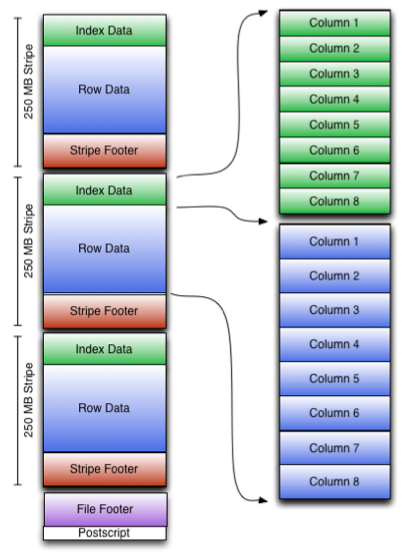


图6-11 Orc格式

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

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

    3）Stripe Footer：存的是各个Stream的类型，长度等信息。

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

### Parquet格式

Parquet是面向分析型业务的列式存储格式，由Twitter和Cloudera合作开发，2015年5月从Apache的孵化器里毕业成为Apache顶级项目。

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

通常情况下，在存储Parquet数据的时候会按照Block大小设置行组的大小，由于一般情况下每一个Mapper任务处理数据的最小单位是一个Block，这样可以把每一个行组由一个Mapper任务处理，增大任务执行并行度。Parquet文件的格式如图6-12所示。

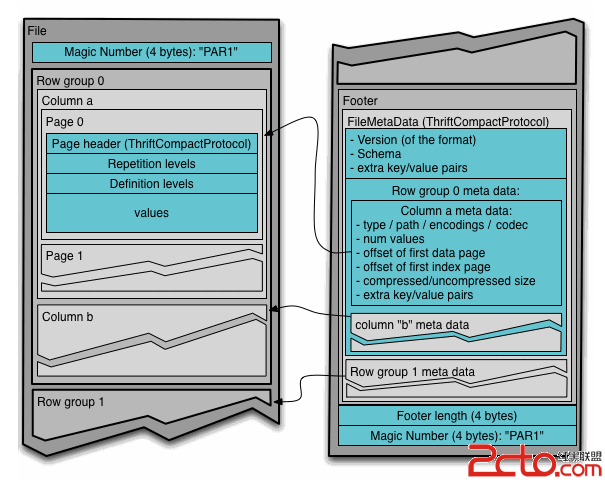


图6-12 Parquet格式

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

### 主流文件存储格式对比实验

从存储文件的压缩比和查询速度两个角度对比。

**存储文件的压缩比测试：**

1. 测试数据



2．TextFile

（1）创建表，存储数据格式为TEXTFILE

|  |
| --- |
| create table log\_text (  track\_time string,  url string,  session\_id string,  referer string,  ip string,  end\_user\_id string,  city\_id string  )  row format delimited fields terminated by '\t'  stored as textfile ; |

（2）向表中加载数据

|  |
| --- |
| hive (default)> load data local inpath '/data/log.data' into table log\_text ; |

（3）查看表中数据大小

|  |
| --- |
| hive (default)> dfs -du -h /user/hive/warehouse/log\_text; |

18.1 M /user/hive/warehouse/log\_text/log.data

3．ORC

（1）创建表，存储数据格式为ORC

|  |
| --- |
| create table log\_orc(  track\_time string,  url string,  session\_id string,  referer string,  ip string,  end\_user\_id string,  city\_id string  )  row format delimited fields terminated by '\t'  stored as orc ; |

（2）向表中加载数据

|  |
| --- |
| hive (default)> insert into table log\_orc select \* from log\_text ; |

（3）查看表中数据大小

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

2.8 M /user/hive/warehouse/log\_orc/000000\_0

4．Parquet

（1）创建表，存储数据格式为parquet

|  |
| --- |
| create table log\_parquet(  track\_time string,  url string,  session\_id string,  referer string,  ip string,  end\_user\_id string,  city\_id string  )  row format delimited fields terminated by '\t'  stored as parquet ; |

（2）向表中加载数据

|  |
| --- |
| hive (default)> insert into table log\_parquet select \* from log\_text ; |

（3）查看表中数据大小

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

13.1 M /user/hive/warehouse/log\_parquet/000000\_0

存储文件的压缩比总结：

ORC > Parquet > textFile

**存储文件的查询速度测试：**

1．TextFile

hive (default)> select count(\*) from log\_text;

\_c0

100000

Time taken: 21.54 seconds, Fetched: 1 row(s)

Time taken: 21.08 seconds, Fetched: 1 row(s)

Time taken: 19.298 seconds, Fetched: 1 row(s)

2．ORC

hive (default)> select count(\*) from log\_orc;

\_c0

100000

Time taken: 20.867 seconds, Fetched: 1 row(s)

Time taken: 22.667 seconds, Fetched: 1 row(s)

Time taken: 18.36 seconds, Fetched: 1 row(s)

3．Parquet

hive (default)> select count(\*) from log\_parquet;

\_c0

100000

Time taken: 22.922 seconds, Fetched: 1 row(s)

Time taken: 21.074 seconds, Fetched: 1 row(s)

Time taken: 18.384 seconds, Fetched: 1 row(s)

存储文件的查询速度总结：查询速度相近。

## 存储和压缩结合

### 修改Hadoop集群具有Snappy压缩方式

1．查看hadoop checknative命令使用

[doit@hadoop104 hadoop-2.7.2]$ hadoop

checknative [-a|-h] check native hadoop and compression libraries availability

2．查看hadoop支持的压缩方式

[doit@hadoop104 hadoop-2.7.2]$ hadoop checknative

17/12/24 20:32:52 WARN bzip2.Bzip2Factory: Failed to load/initialize native-bzip2 library system-native, will use pure-Java version

17/12/24 20:32:52 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library

Native library checking:

hadoop: true /opt/module/hadoop-2.7.2/lib/native/libhadoop.so

zlib: true /lib64/libz.so.1

snappy: false

lz4: true revision:99

bzip2: false

3．将编译好的支持Snappy压缩的hadoop-2.7.2.tar.gz包导入到hadoop102的/opt/software中

4．解压hadoop-2.7.2.tar.gz到当前路径

[doit@hadoop102 software]$ tar -zxvf hadoop-2.7.2.tar.gz

5．进入到/opt/software/hadoop-2.7.2/lib/native路径可以看到支持Snappy压缩的动态链接库

[doit@hadoop102 native]$ pwd

/opt/software/hadoop-2.7.2/lib/native

[doit@hadoop102 native]$ ll

-rw-r--r--. 1 doit doit 472950 9月 1 10:19 libsnappy.a

-rwxr-xr-x. 1 doit doit 955 9月 1 10:19 libsnappy.la

lrwxrwxrwx. 1 doit doit 18 12月 24 20:39 libsnappy.so -> libsnappy.so.1.3.0

lrwxrwxrwx. 1 doit doit 18 12月 24 20:39 libsnappy.so.1 -> libsnappy.so.1.3.0

-rwxr-xr-x. 1 doit doit 228177 9月 1 10:19 libsnappy.so.1.3.0

6．拷贝/opt/software/hadoop-2.7.2/lib/native里面的所有内容到开发集群的/opt/module/hadoop-2.7.2/lib/native路径上

[doit@hadoop102 native]$ cp ../native/\* /opt/module/hadoop-2.7.2/lib/native/

7．分发集群

[doit@hadoop102 lib]$ xsync native/

8．再次查看hadoop支持的压缩类型

[doit@hadoop102 hadoop-2.7.2]$ hadoop checknative

17/12/24 20:45:02 WARN bzip2.Bzip2Factory: Failed to load/initialize native-bzip2 library system-native, will use pure-Java version

17/12/24 20:45:02 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib library

Native library checking:

hadoop: true /opt/module/hadoop-2.7.2/lib/native/libhadoop.so

zlib: true /lib64/libz.so.1

snappy: true /opt/module/hadoop-2.7.2/lib/native/libsnappy.so.1

lz4: true revision:99

bzip2: false

9.重新启动hadoop集群和hive

### 测试存储和压缩

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

ORC存储方式的压缩：

表6-12

|  |  |  |
| --- | --- | --- |
| Key | Default | Notes |
| orc.compress | ZLIB | high level compression (one of NONE, ZLIB, SNAPPY) |
| orc.compress.size | 262,144 | number of bytes in each compression chunk |
| orc.stripe.size | 67,108,864 | number of bytes in each stripe |
| orc.row.index.stride | 10,000 | number of rows between index entries (must be >= 1000) |
| orc.create.index | true | whether to create row indexes |
| orc.bloom.filter.columns | "" | comma separated list of column names for which bloom filter should be created |
| orc.bloom.filter.fpp | 0.05 | false positive probability for bloom filter (must >0.0 and <1.0) |

1．创建一个非压缩的的ORC存储方式

（1）建表语句

|  |
| --- |
| create table log\_orc\_none(  track\_time string,  url string,  session\_id string,  referer string,  ip string,  end\_user\_id string,  city\_id string  )  row format delimited fields terminated by '\t'  stored as orc tblproperties ("orc.compress"="NONE"); |

（2）插入数据

|  |
| --- |
| hive (default)> insert into table log\_orc\_none select \* from log\_text ; |

（3）查看插入后数据

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

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

2．创建一个SNAPPY压缩的ORC存储方式

（1）建表语句

|  |
| --- |
| create table log\_orc\_snappy(  track\_time string,  url string,  session\_id string,  referer string,  ip string,  end\_user\_id string,  city\_id string  )  row format delimited fields terminated by '\t'  stored as orc tblproperties ("orc.compress"="SNAPPY"); |

（2）插入数据

|  |
| --- |
| hive (default)> insert into table log\_orc\_snappy select \* from log\_text ; |

（3）查看插入后数据

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

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

3．上一节中默认创建的ORC存储方式，导入数据后的大小为

2.8 M /user/hive/warehouse/log\_orc/000000\_0

比Snappy压缩的还小。原因是orc存储文件默认采用ZLIB压缩。比snappy压缩的小。

4．存储方式和压缩总结

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

# 企业级调优

## Fetch抓取

Fetch抓取是指，Hive中对某些情况的查询可以不必使用MapReduce计算。例如：SELECT \* FROM employees;在这种情况下，Hive可以简单地读取employee对应的存储目录下的文件，然后输出查询结果到控制台。

在hive-default.xml.template文件中hive.fetch.task.conversion默认是more，老版本hive默认是minimal，该属性修改为more以后，在全局查找、字段查找、limit查找等都不走mapreduce。

|  |
| --- |
| <property>  <name>hive.fetch.task.conversion</name>  <value>more</value>  <description>  Expects one of [none, minimal, more].  Some select queries can be converted to single FETCH task minimizing latency.  Currently the query should be single sourced not having any subquery and should not have  any aggregations or distincts (which incurs RS), lateral views and joins.  0. none : disable hive.fetch.task.conversion  1. minimal : SELECT STAR, FILTER on partition columns, LIMIT only  2. more : SELECT, FILTER, LIMIT only (support TABLESAMPLE and virtual columns)  </description>  </property> |

案例实操：

1）把hive.fetch.task.conversion设置成none，然后执行查询语句，都会执行mapreduce程序。

hive (default)> set hive.fetch.task.conversion=none;

hive (default)> select \* from emp;

hive (default)> select ename from emp;

hive (default)> select ename from emp limit 3;

2）把hive.fetch.task.conversion设置成more，然后执行查询语句，如下查询方式都不会执行mapreduce程序。

hive (default)> set hive.fetch.task.conversion=more;

hive (default)> select \* from emp;

hive (default)> select ename from emp;

hive (default)> select ename from emp limit 3;

## 本地模式

大多数的Hadoop Job是需要Hadoop提供的完整的可扩展性来处理大数据集的。不过，有时Hive的输入数据量是非常小的。在这种情况下，为查询触发执行任务消耗的时间可能会比实际job的执行时间要多的多。对于大多数这种情况，Hive可以通过本地模式在单台机器上处理所有的任务。对于小数据集，执行时间可以明显被缩短。

用户可以通过设置hive.exec.mode.local.auto的值为true，来让Hive在适当时候自动启动这个优化。

set hive.exec.mode.local.auto=true;  //开启本地mr

//设置local mr的最大输入数据量，当输入数据量小于这个值时采用local  mr的方式，默认为134217728，即128M

set hive.exec.mode.local.auto.inputbytes.max=50000000;

//设置local mr的最大输入文件个数，当输入文件个数小于这个值时采用local mr的方式，默认为4

set hive.exec.mode.local.auto.input.files.max=10;

案例实操：

1）开启本地模式，并执行查询语句

hive (default)> set hive.exec.mode.local.auto=true;

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

Time taken: 1.328 seconds, Fetched: 14 row(s)

2）关闭本地模式，并执行查询语句

hive (default)> set hive.exec.mode.local.auto=false;

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

Time taken: 20.09 seconds, Fetched: 14 row(s)

## 表的优化

### 小表、大表Join

尽量的将小表放在join的前面

将key相对分散，并且数据量小的表放在join的左边，这样可以有效减少内存溢出错误发生的几率；再进一步，可以使用map join让小的维度表（1000条以下的记录条数）先进内存。在map端完成reduce。

实际测试发现：新版的hive已经对小表JOIN大表和大表JOIN小表进行了优化。小表放在左边和右边已经没有明显区别。

**案例实操**

1. 需求

测试大表JOIN小表和小表JOIN大表的效率

2．建大表、小表和JOIN后表的语句

|  |
| --- |
| // 创建大表  create table bigtable(id bigint, time bigint, uid string, keyword string, url\_rank int, click\_num int, click\_url string) row format delimited fields terminated by '\t';  // 创建小表  create table smalltable(id bigint, time bigint, uid string, keyword string, url\_rank int, click\_num int, click\_url string) row format delimited fields terminated by '\t';  // 创建join后表的语句  create table jointable(id bigint, time bigint, uid string, keyword string, url\_rank int, click\_num int, click\_url string) row format delimited fields terminated by '\t'; |

3．分别向大表和小表中导入数据

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

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

4．关闭mapjoin功能（默认是打开的）

set hive.auto.convert.join = false;

5．执行小表JOIN大表语句

insert overwrite table jointable

select b.id, b.time, b.uid, b.keyword, b.url\_rank, b.click\_num, b.click\_url

from smalltable s

left join bigtable b

on b.id = s.id;

Time taken: 35.921 seconds

No rows affected (44.456 seconds)

6．执行大表JOIN小表语句

insert overwrite table jointable

select b.id, b.time, b.uid, b.keyword, b.url\_rank, b.click\_num, b.click\_url

from bigtable b

left join smalltable s

on s.id = b.id;

Time taken: 34.196 seconds

No rows affected (26.287 seconds)

### 大表Join大表

1．空KEY过滤

有时join超时是因为某些key对应的数据太多，而相同key对应的数据都会发送到相同的reducer上，从而导致内存不够。此时我们应该仔细分析这些异常的key，很多情况下，这些key对应的数据是异常数据，我们需要在SQL语句中进行过滤。例如key对应的字段为空，操作如下：

案例实操

（1）配置历史服务器

配置mapred-site.xml

|  |
| --- |
| <property>  <name>mapreduce.jobhistory.address</name>  <value>hadoop102:10020</value>  </property>  <property>  <name>mapreduce.jobhistory.webapp.address</name>  <value>hadoop102:19888</value>  </property> |

启动历史服务器

sbin/mr-jobhistory-daemon.sh start historyserver

查看jobhistory

[http://doit01:19888/jobhistory](http://hadoop102:19888/jobhistory)

（2）创建原始数据表、空id表、合并后数据表

|  |
| --- |
| // 创建原始表  create table ori(id bigint, time bigint, uid string, keyword string, url\_rank int, click\_num int, click\_url string) row format delimited fields terminated by '\t';  // 创建空id表  create table nullidtable(id bigint, time bigint, uid string, keyword string, url\_rank int, click\_num int, click\_url string) row format delimited fields terminated by '\t';  // 创建join后表的语句  create table jointable(id bigint, time bigint, uid string, keyword string, url\_rank int, click\_num int, click\_url string) row format delimited fields terminated by '\t'; |

（3）分别加载原始数据和空id数据到对应表中

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

hive (default)> load data local inpath '/opt/module/datas/nullid' into table nullidtable;

（4）测试不过滤空id

hive (default)> insert overwrite table jointable

select n.\* from nullidtable n left join ori o on n.id = o.id;

Time taken: 42.038 seconds

Time taken: 37.284 seconds

（5）测试过滤空id

hive (default)> insert overwrite table jointable

select n.\* from (select \* from nullidtable where id is not null ) n left join ori o on n.id = o.id;

Time taken: 31.725 seconds

Time taken: 28.876 seconds

2．空key转换

有时虽然某个key为空对应的数据很多，但是相应的数据不是异常数据，必须要包含在join的结果中，此时我们可以表a中key为空的字段赋一个随机的值，使得数据随机均匀地分不到不同的reducer上。例如：

**案例实操：**

**不随机分布空null值：**

（1）设置5个reduce个数

set mapreduce.job.reduces = 5;

（2）JOIN两张表

insert overwrite table jointable

select n.\* from nullidtable n left join ori b on n.id = b.id;

**结果：如图6-13所示，可以看出来，出现了数据倾斜，某些reducer的资源消耗远大于其他reducer。**

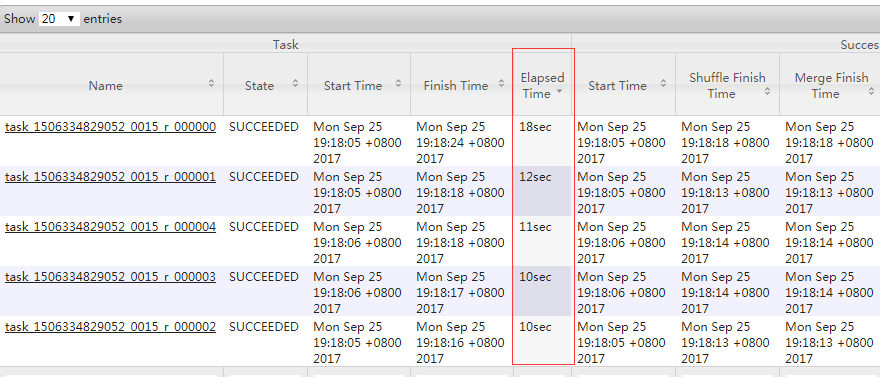


图6-13 空key转换

随机分布空null值

（1）设置5个reduce个数

set mapreduce.job.reduces = 5;

（2）JOIN两张表

insert overwrite table jointable

select n.\* from nullidtable n full join ori o on

case when n.id is null then concat('hive', rand()) else n.id end = o.id;

**结果：如图6-14所示，可以看出来，消除了数据倾斜，负载均衡reducer的资源消耗**

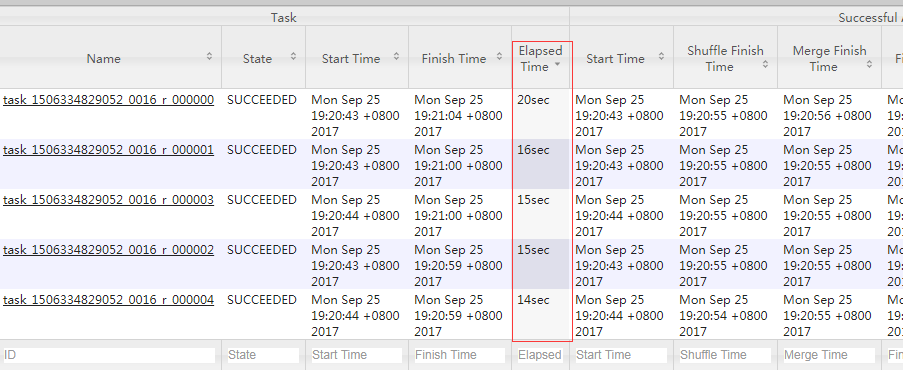


图6-14 随机分布空值

### Count(Distinct) 去重统计

数据量小的时候无所谓，数据量大的情况下，由于COUNT DISTINCT操作需要用一个Reduce Task来完成，这一个Reduce需要处理的数据量太大，就会导致整个Job很难完成，一般COUNT DISTINCT使用先GROUP BY再COUNT的方式替换：

**案例实操**

1. 创建一张大表

hive (default)> create table bigtable(id bigint, time bigint, uid string, keyword

string, url\_rank int, click\_num int, click\_url string) row format delimited

fields terminated by '\t';

2．加载数据

hive (default)> load data local inpath '/opt/module/datas/bigtable' into table

bigtable;

3．设置5个reduce个数

set mapreduce.job.reduces = 5;

4．执行去重id查询

hive (default)> select count(distinct id) from bigtable;

Stage-Stage-1: Map: 1 Reduce: 1 Cumulative CPU: 7.12 sec HDFS Read: 120741990 HDFS Write: 7 SUCCESS

Total MapReduce CPU Time Spent: 7 seconds 120 msec

OK

c0

100001

Time taken: 23.607 seconds, Fetched: 1 row(s)

5．采用GROUP by去重id

hive (default)> select count(id) from (select id from bigtable group by id) a;

Stage-Stage-1: Map: 1 Reduce: 5 Cumulative CPU: 17.53 sec HDFS Read: 120752703 HDFS Write: 580 SUCCESS

Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 4.29 sec HDFS Read: 9409 HDFS Write: 7 SUCCESS

Total MapReduce CPU Time Spent: 21 seconds 820 msec

OK

\_c0

100001

Time taken: 50.795 seconds, Fetched: 1 row(s)

虽然会多用一个Job来完成，但在数据量大的情况下，这个绝对是值得的。

### 笛卡尔积

尽量避免笛卡尔积，join的时候不加on条件，或者无效的on条件，Hive只能使用1个reducer来完成笛卡尔积。

### 行列过滤

列处理：在SELECT中，只拿需要的列，如果有，尽量使用分区过滤，少用SELECT \*。

行处理：在分区剪裁中，当使用外关联时，如果将副表的过滤条件写在Where后面，那么就会先全表关联，之后再过滤，比如：

**案例实操：**

1．测试先关联两张表，再用where条件过滤

hive (default)> select o.id from bigtable b

join ori o on o.id = b.id

where o.id <= 10;

Time taken: 34.406 seconds, Fetched: 100 row(s)

2．通过子查询后，再关联表

hive (default)> select b.id from bigtable b

join (select id from ori where id <= 10 ) o on b.id = o.id;

Time taken: 30.058 seconds, Fetched: 100 row(s)

### 动态分区调整

分区 分桶

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

1．开启动态分区参数设置

（1）开启动态分区功能（默认true，开启）

hive.exec.dynamic.partition=true

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

hive.exec.dynamic.partition.mode=nonstrict

（3）在所有执行MR的节点上，最大一共可以创建多少个动态分区。

hive.exec.max.dynamic.partitions=1000

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

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

（5）整个MR Job中，最大可以创建多少个HDFS文件。

hive.exec.max.created.files=100000

（6）当有空分区生成时，是否抛出异常。一般不需要设置。

hive.error.on.empty.partition=false

2．案例实操

需求：将ori中的数据按照时间(如：20111230000008)，插入到目标表ori\_partitioned\_target的相应分区中。

（1）创建分区表

|  |
| --- |
| create table ori\_partitioned(id bigint, time bigint, uid string, keyword string,  url\_rank int, click\_num int, click\_url string)  partitioned by (p\_time bigint)  row format delimited fields terminated by '\t'; |

（2）加载数据到分区表中

|  |
| --- |
| hive (default)> load data local inpath '/home/doit/ds1' into table  ori\_partitioned partition(p\_time='20111230000010') ;  hive (default)> load data local inpath '/home/doit/ds2' into table ori\_partitioned partition(p\_time='20111230000011') ; |

（3）创建目标分区表

|  |
| --- |
| create table ori\_partitioned\_target(id bigint, time bigint, uid string,  keyword string, url\_rank int, click\_num int, click\_url string) PARTITIONED BY (p\_time STRING) row format delimited fields terminated by '\t'; |

（4）设置动态分区

|  |
| --- |
| set hive.exec.dynamic.partition = true;  set hive.exec.dynamic.partition.mode = nonstrict;  set hive.exec.max.dynamic.partitions = 1000;  set hive.exec.max.dynamic.partitions.pernode = 100;  set hive.exec.max.created.files = 100000;  set hive.error.on.empty.partition = false;  hive (default)> insert overwrite table ori\_partitioned\_target partition (p\_time)  select id, time, uid, keyword, url\_rank, click\_num, click\_url, p\_time from ori\_partitioned; |

（5）查看目标分区表的分区情况

hive (default)> show partitions ori\_partitioned\_target;

### 分桶

详见6.6章。

### 分区

详见4.6章。

## 数据倾斜

### MapJoin

如果不指定MapJoin或者不符合MapJoin的条件，那么Hive解析器会将Join操作转换成Common Join，即：在Reduce阶段完成join。容易发生数据倾斜。可以用MapJoin把小表全部加载到内存在map端进行join，避免reducer处理。

1．开启MapJoin参数设置

（1）设置自动选择Mapjoin

set hive.auto.convert.join = true; 默认为true

（2）大表小表的阈值设置（默认25M一下认为是小表）：

set hive.mapjoin.smalltable.filesize=25000000;

2．MapJoin工作机制，如图6-15所示

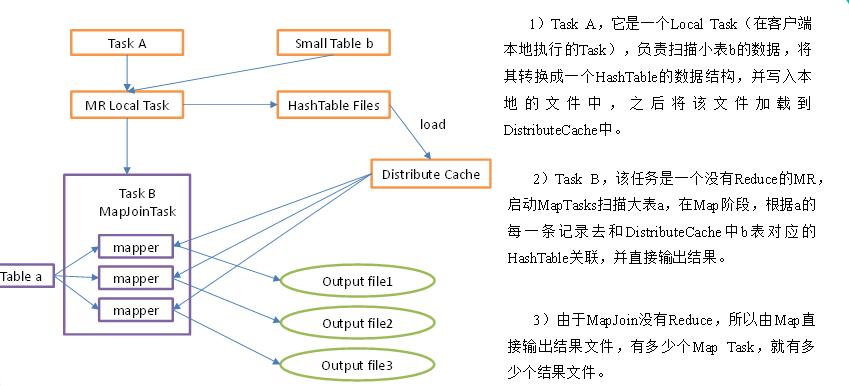


图6-15 MapJoin工作机制

**案例实操：**

（1）开启Mapjoin功能

set hive.auto.convert.join = true; 默认为true

（2）执行小表JOIN大表语句

insert overwrite table jointable

select b.id, b.time, b.uid, b.keyword, b.url\_rank, b.click\_num, b.click\_url

from smalltable s

join bigtable b

on s.id = b.id;

Time taken: 24.594 seconds

（3）执行大表JOIN小表语句insert overwrite table jointable

select b.id, b.time, b.uid, b.keyword, b.url\_rank, b.click\_num, b.click\_url

from bigtable b

join smalltable s

on s.id = b.id;

Time taken: 24.315 seconds

### Group By

默认情况下，Map阶段同一Key数据分发给一个reduce，当一个key数据过大时就倾斜了。

并不是所有的聚合操作都需要在Reduce端完成，很多聚合操作都可以先在Map端进行部分聚合，最后在Reduce端得出最终结果。

1．开启Map端聚合参数设置

（1）是否在Map端进行聚合，默认为True

hive.map.aggr = true

（2）在Map端进行聚合操作的条目数目

hive.groupby.mapaggr.checkinterval = 100000

（3）有数据倾斜的时候进行负载均衡（默认是false）

hive.groupby.skewindata = true

当选项设定为 true，生成的查询计划会有两个MR Job。第一个MR Job中，Map的输出结果会随机分布到Reduce中，每个Reduce做部分聚合操作，并输出结果，这样处理的结果是相同的Group By Key有可能被分发到不同的Reduce中，从而达到负载均衡的目的；第二个MR Job再根据预处理的数据结果按照Group By Key分布到Reduce中（这个过程可以保证相同的Group By Key被分布到同一个Reduce中），最后完成最终的聚合操作。

### 合理设置Map数

**1）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。**

主要的决定因素有：input的文件总个数，input的文件大小，集群设置的文件块大小。

**2）是不是map数越多越好？**

答案是否定的。如果一个任务有很多小文件（远远小于块大小128m），则每个小文件也会被当做一个块，用一个map任务来完成，而一个map任务启动和初始化的时间远远大于逻辑处理的时间，就会造成很大的资源浪费。而且，同时可执行的map数是受限的。

**3）是不是保证每个map处理接近128m的文件块，就高枕无忧了？**

答案也是不一定。比如有一个127m的文件，正常会用一个map去完成，但这个文件只有一个或者两个小字段，却有几千万的记录，如果map处理的逻辑比较复杂，用一个map任务去做，肯定也比较耗时。

针对上面的问题2和3，我们需要采取两种方式来解决：即减少map数和增加map数；

### 小文件进行合并

在map执行前合并小文件，减少map数：CombineHiveInputFormat具有对小文件进行合并的功能（系统默认的格式）。HiveInputFormat没有对小文件合并功能。

set hive.input.format= org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;

### 复杂文件增加Map数 128M

当input的文件都很大，任务逻辑复杂，map执行非常慢的时候，可以考虑增加Map数，来使得每个map处理的数据量减少，从而提高任务的执行效率。

增加map的方法为：根据computeSliteSize(Math.max(minSize,Math.min(maxSize,blocksize)))=blocksize=128M公式，调整maxSize最大值。让maxSize最大值低于blocksize就可以增加map的个数。

**案例实操：**

1．执行查询

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

Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1

2．设置最大切片值为100个字节

hive (default)> set mapreduce.input.fileinputformat.split.maxsize=100;

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

Hadoop job information for Stage-1: number of mappers: 6; number of reducers: 1

### 合理设置Reduce数

1．调整reduce个数方法一

（1）每个Reduce处理的数据量默认是256MB

hive.exec.reducers.bytes.per.reducer=256000000

（2）每个任务最大的reduce数，默认为1009

hive.exec.reducers.max=1009

（3）计算reducer数的公式

N=min(参数2，总输入数据量/参数1)

2．调整reduce个数方法二

在hadoop的mapred-default.xml文件中修改

设置每个job的Reduce个数

set mapreduce.job.reduces = 15;

3．reduce个数并不是越多越好

1）过多的启动和初始化reduce也会消耗时间和资源；

2）另外，有多少个reduce，就会有多少个输出文件，如果生成了很多个小文件，那么如果这些小文件作为下一个任务的输入，则也会出现小文件过多的问题；

在设置reduce个数的时候也需要考虑这两个原则：处理大数据量利用合适的reduce数；使单个reduce任务处理数据量大小要合适；

## 并行执行

Hive会将一个查询转化成一个或者多个阶段。这样的阶段可以是MapReduce阶段、抽样阶段、合并阶段、limit阶段。或者Hive执行过程中可能需要的其他阶段。默认情况下，Hive一次只会执行一个阶段。不过，某个特定的job可能包含众多的阶段，而这些阶段可能并非完全互相依赖的，也就是说有些阶段是可以并行执行的，这样可能使得整个job的执行时间缩短。不过，如果有更多的阶段可以并行执行，那么job可能就越快完成。

通过设置参数hive.exec.parallel值为true，就可以开启并发执行。不过，在共享集群中，需要注意下，如果job中并行阶段增多，那么集群利用率就会增加。

set hive.exec.parallel=true;   //打开任务并行执行

set hive.exec.parallel.thread.number=16; //同一个sql允许最大并行度，默认为8。

当然，得是在系统资源比较空闲的时候才有优势，否则，没资源，并行也起不来。

## 严格模式

Hive提供了一个严格模式，可以防止用户执行那些可能意想不到的不好的影响的查询。

通过设置属性hive.mapred.mode值为默认是非严格模式nonstrict 。开启严格模式需要修改hive.mapred.mode值为strict，开启严格模式可以禁止3种类型的查询。

<property>

<name>hive.mapred.mode</name>

<value>strict</value>

<description>

The mode in which the Hive operations are being performed.

In strict mode, some risky queries are not allowed to run. They include:

Cartesian Product.

No partition being picked up for a query.

Comparing bigints and strings.

Comparing bigints and doubles.

Orderby without limit.

</description>

</property>

1. 对于分区表，除非where语句中含有分区字段过滤条件来限制范围，否则不允许执行。换句话说，就是用户不允许扫描所有分区。进行这个限制的原因是，通常分区表都拥有非常大的数据集，而且数据增加迅速。没有进行分区限制的查询可能会消耗令人不可接受的巨大资源来处理这个表。
2. 对于使用了order by语句的查询，要求必须使用limit语句。因为order by为了执行排序过程会将所有的结果数据分发到同一个Reducer中进行处理，强制要求用户增加这个LIMIT语句可以防止Reducer额外执行很长一段时间。
3. 限制笛卡尔积的查询。对关系型数据库非常了解的用户可能期望在执行JOIN查询的时候不使用ON语句而是使用where语句，这样关系数据库的执行优化器就可以高效地将WHERE语句转化成那个ON语句。不幸的是，Hive并不会执行这种优化，因此，如果表足够大，那么这个查询就会出现不可控的情况。

## JVM重用

JVM重用是Hadoop调优参数的内容，其对Hive的性能具有非常大的影响，特别是对于很难避免小文件的场景或task特别多的场景，这类场景大多数执行时间都很短。

Hadoop的默认配置通常是使用派生JVM来执行map和Reduce任务的。这时JVM的启动过程可能会造成相当大的开销，尤其是执行的job包含有成百上千task任务的情况。JVM重用可以使得JVM实例在同一个job中重新使用N次。N的值可以在Hadoop的mapred-site.xml文件中进行配置。通常在10-20之间，具体多少需要根据具体业务场景测试得出。

<property>

<name>mapreduce.job.jvm.numtasks</name>

<value>10</value>

<description>How many tasks to run per jvm. If set to -1, there is

no limit.

</description>

</property>

这个功能的缺点是，开启JVM重用将一直占用使用到的task插槽，以便进行重用，直到任务完成后才能释放。如果某个“不平衡的”job中有某几个reduce task执行的时间要比其他Reduce task消耗的时间多的多的话，那么保留的插槽就会一直空闲着却无法被其他的job使用，直到所有的task都结束了才会释放。

## 推测执行

在分布式集群环境下，因为程序Bug（包括Hadoop本身的bug），负载不均衡或者资源分布不均等原因，会造成同一个作业的多个任务之间运行速度不一致，有些任务的运行速度可能明显慢于其他任务（比如一个作业的某个任务进度只有50%，而其他所有任务已经运行完毕），则这些任务会拖慢作业的整体执行进度。为了避免这种情况发生，Hadoop采用了推测执行（Speculative Execution）机制，它根据一定的法则推测出“拖后腿”的任务，并为这样的任务启动一个备份任务，让该任务与原始任务同时处理同一份数据，并最终选用最先成功运行完成任务的计算结果作为最终结果。

设置开启推测执行参数：Hadoop的mapred-site.xml文件中进行配置

|  |
| --- |
| <property>  <name>mapreduce.map.speculative</name>  <value>true</value>  <description>If true, then multiple instances of some map tasks  may be executed in parallel.</description>  </property>  <property>  <name>mapreduce.reduce.speculative</name>  <value>true</value>  <description>If true, then multiple instances of some reduce tasks  may be executed in parallel.</description>  </property> |

不过hive本身也提供了配置项来控制reduce-side的推测执行：

|  |
| --- |
| <property>  <name>hive.mapred.reduce.tasks.speculative.execution</name>  <value>true</value>  <description>Whether speculative execution for reducers should be turned on. </description>  </property> |

关于调优这些推测执行变量，还很难给一个具体的建议。如果用户对于运行时的偏差非常敏感的话，那么可以将这些功能关闭掉。如果用户因为输入数据量很大而需要执行长时间的map或者Reduce task的话，那么启动推测执行造成的浪费是非常巨大大。

## 压缩

详见第8章。

## 执行计划（Explain）

1．基本语法

EXPLAIN [EXTENDED | DEPENDENCY | AUTHORIZATION] query

2．案例实操

（1）查看下面这条语句的执行计划

hive (default)> explain select \* from emp;

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

（2）查看详细执行计划

hive (default)> explain extended select \* from emp;

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

# JDBC

jdbc是jdk开发的操作数据库的一套标准API(接口方法)

基本的开发步骤:

1 创建项目 导入jar包

2 注册驱动

3 获取连接

4 获取执行SQL语句的对象

5 sql 发送执行sql

6 获取结果数据

7 释放资源

|  |
| --- |
| <dependencies>  <dependency>  <groupId>org.apache.hadoop</groupId>  <artifactId>hadoop-common</artifactId>  <version>2.8.5</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-client -->  <dependency>  <groupId>org.apache.hadoop</groupId>  <artifactId>hadoop-client</artifactId>  <version>2.8.5</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.hive/hive-jdbc  数据库的驱动 连接的是何种数据库 -->  <dependency>  <groupId>org.apache.hive</groupId>  <artifactId>hive-jdbc</artifactId>  <version>2.3.5</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.hive/hive-metastore -->  <dependency>  <groupId>org.apache.hive</groupId>  <artifactId>hive-metastore</artifactId>  <version>2.3.5</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.hive/hive-metastore -->  <dependency>  <groupId>org.apache.hive</groupId>  <artifactId>hive-exec</artifactId>  <version>2.3.5</version>  </dependency>  </dependencies> |

## 创建maven项目添加依赖

**hive开启hiveserver2 服务 ; 开启hive的远程连接的服务**

**service mysqld start**

## JAVAAPI

|  |
| --- |
| public class App {  private static String driverName = "org.apache.hive.jdbc.HiveDriver";  public static void main(String[] args) throws SQLException {  try {  Class.forName(driverName);  } catch (ClassNotFoundException e) {  e.printStackTrace();  System.exit(1);  }  Connection con = DriverManager.getConnection("jdbc:hive2://doit01:10000/demo", "root", "root");  Statement stmt = con.createStatement();  String sql = "select \* from tb\_case";  ResultSet res = stmt.executeQuery(sql);  if (res.next()) {  System.out.println(res.getString(1));  }  }  } |

# 整合HBASE

### HBase与Hive的对比

1．Hive

(1) 数据仓库

Hive的本质其实就相当于将HDFS中已经存储的文件在Mysql中做了一个双射关系，以方便使用HQL去管理查询。

(2) 用于数据分析、清洗

Hive适用于离线的数据分析和清洗，延迟较高。

(3) 基于HDFS、MapReduce

Hive存储的数据依旧在DataNode上，编写的HQL语句终将是转换为MapReduce代码执行。

2．HBase

(1) 数据库

是一种面向列存储的非关系型数据库。

(2) 用于存储结构化和非结构化的数据

适用于单表非关系型数据的存储，不适合做关联查询，类似JOIN等操作。

(3) 基于HDFS

数据持久化存储的体现形式是Hfile，存放于DataNode中，被ResionServer以region的形式进行管理。

(4) 延迟较低，接入在线业务使用

面对大量的企业数据，HBase可以直线单表大量数据的存储，同时提供了高效的数据访问速度。

### HBase与Hive集成使用

尖叫提示：HBase与Hive的集成在最新的两个版本中无法兼容。所以，我们只能含着泪勇敢的重新编译：hive-hbase-handler-1.2.2.jar！！好气！！

**环境准备**

因为我们后续可能会在操作Hive的同时对HBase也会产生影响，所以Hive需要持有操作HBase的Jar，那么接下来拷贝Hive所依赖的Jar包（或者使用软连接的形式）。

|  |
| --- |
| export HBASE\_HOME=/opt/module/hbase  export HIVE\_HOME=/opt/module/hive  ln -s $HBASE\_HOME/lib/hbase-common-1.3.1.jar $HIVE\_HOME/lib/hbase-common-1.3.1.jar  ln -s $HBASE\_HOME/lib/hbase-server-1.3.1.jar $HIVE\_HOME/lib/hbase-server-1.3.1.jar  ln -s $HBASE\_HOME/lib/hbase-client-1.3.1.jar $HIVE\_HOME/lib/hbase-client-1.3.1.jar  ln -s $HBASE\_HOME/lib/hbase-protocol-1.3.1.jar $HIVE\_HOME/lib/hbase-protocol-1.3.1.jar  ln -s $HBASE\_HOME/lib/hbase-it-1.3.1.jar $HIVE\_HOME/lib/hbase-it-1.3.1.jar  ln -s $HBASE\_HOME/lib/htrace-core-3.1.0-incubating.jar $HIVE\_HOME/lib/htrace-core-3.1.0-incubating.jar  ln -s $HBASE\_HOME/lib/hbase-hadoop2-compat-1.3.1.jar $HIVE\_HOME/lib/hbase-hadoop2-compat-1.3.1.jar  ln -s $HBASE\_HOME/lib/hbase-hadoop-compat-1.3.1.jar $HIVE\_HOME/lib/hbase-hadoop-compat-1.3.1.jar |

同时在hive-site.xml中修改zookeeper的属性，如下：

|  |
| --- |
| <property>  <name>hive.zookeeper.quorum</name>  <value>linux01,linux02,linux03</value>  <description>The list of ZooKeeper servers to talk to. This is only needed for read/write locks.</description>  </property>  <property>  <name>hive.zookeeper.client.port</name>  <value>2181</value>  <description>The port of ZooKeeper servers to talk to. This is only needed for read/write locks.</description>  </property> |

1．案例一

**目标：**建立Hive表，关联HBase表，插入数据到Hive表的同时能够影响HBase表。

**分步实现：**

(1) 在Hive中创建表同时关联HBase

|  |
| --- |
|  |

提示：完成之后，可以分别进入Hive和HBase查看，都生成了对应的表

(2) 在Hive中创建临时中间表，用于load文件中的数据

提示：不能将数据直接load进Hive所关联HBase的那张表中

CREATE TABLE emp(

empno int,

ename string,

job string,

mgr int,

hiredate string,

sal double,

comm double,

deptno int)

row format delimited fields terminated by '\t';

(3) 向Hive中间表中load数据

|  |
| --- |
| hive> load data local inpath '/home/admin/softwares/data/emp.txt' into table emp; |

(4) 通过insert命令将中间表中的数据导入到Hive关联HBase的那张表中

|  |
| --- |
| hive> insert into table hive\_hbase\_emp\_table select \* from emp; |

(5) 查看Hive以及关联的HBase表中是否已经成功的同步插入了数据

Hive：

|  |
| --- |
| hive> select \* from hive\_hbase\_emp\_table; |

HBase：

|  |
| --- |
| hbase> scan ‘hbase\_emp\_table’ |

2．案例二

**目标：**在HBase中已经存储了某一张表hbase\_emp\_table，然后在Hive中创建一个外部表来关联HBase中的hbase\_emp\_table这张表，使之可以借助Hive来分析HBase这张表中的数据。

**注：**该案例2紧跟案例1的脚步，所以完成此案例前，请先完成案例1。

**分步实现：**

(1) 在Hive中创建外部表

|  |
| --- |
| CREATE EXTERNAL TABLE relevance\_hbase\_emp(  empno int,  ename string,  job string,  mgr int,  hiredate string,  sal double,  comm double,  deptno int)  STORED BY  'org.apache.hadoop.hive.hbase.HBaseStorageHandler'  WITH SERDEPROPERTIES ("hbase.columns.mapping" =  ":key,info:ename,info:job,info:mgr,info:hiredate,info:sal,info:comm,info:deptno")  TBLPROPERTIES ("hbase.table.name" = "hbase\_emp\_table"); |

(2) 关联后就可以使用Hive函数进行一些分析操作了

|  |
| --- |
| hive (default)> select \* from relevance\_hbase\_emp; |

**约束**

唯一约束 unique 保证字段的值唯一

非空约束 not null 保证当前字段不能为空

主键约束 primary key 唯一 非空

默认值约束 default 当前字段的默认值

保证数据的安全性和完整性 (正确)

外键约束 , 保证一条数据的正确性

auto\_increment

冗余

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id / promary key(唯一) | name/约束 not null | age | address unique | sal | role default |
| 1 | lisi | 23 | 北京 | 10000 | 普通用户 |
| 2 |  | 34.5 | 上海 | 9000 | vip |
| 1 | lisi | abc | 北京 | 10000 | 普通用户 |

数据类型： java