**HIVE从入门到精通**

目录

[HIVE介绍 1](#_Toc355269938)

[二、 hive的安装和配置 7](#_Toc355269939)

[三、 hive与hbase集成 13](#_Toc355269940)

[四、 HIVE创建目录和表 15](#_Toc355269941)

[六、 HIVE查询 22](#_Toc355269942)

[七、 HIVE视图 28](#_Toc355269943)

[八、 索引 29](#_Toc355269944)

[九、 hive schema 30](#_Toc355269945)

[十、 Hive join 32](#_Toc355269946)

[十一、 Hive基本语法 36](#_Toc355269947)

[十二、 Hive操作语句 39](#_Toc355269948)

[十三、 数据操作语句 50](#_Toc355269949)

[Hive 优化 55](#_Toc355269950)

# HIVE介绍

主要介绍

背景及体系结构

1背景

应用于工业的商务智能收集分析所需的数据集正在大量增长，使得传统的数据仓库解决方案变得过于昂贵。[Hadoop](http://wiki.apache.org/hadoop) 是一个流行的开源map-reduce实现，用于像yahoo, Facebook一类的公司。来存储和处理商用硬件上的大范围数据集。然而map-reduce程序模型还是处于很低级别，即需要开发者来书写客户程序，这些程序往往难于维护与重用。

用hbase做数据库，但由于hbase没有类sql查询方式,所以操作和计算数据非常不方便，于是整合hive，让hive支撑在hbase数据库层面的hql查询。hive也叫做数据仓库。

2定义

Hive是基于Hadoop(HDFS, MapReduce)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数据库表，并提供类SQL查询功能。

本质是将SQL转换为MapReduce程序。

3体系结构

Hive本身建立在Hadoop的体系结构上，可以将结构化的数据文件映射为一张数据库表，并提供完整的sql查询功能，可以将sql语句转换为MapReduce任务进行。并按照该计划生成MapReduce任务后交给Hadoop集群处理，Hive的体系结构如图1-1所示：

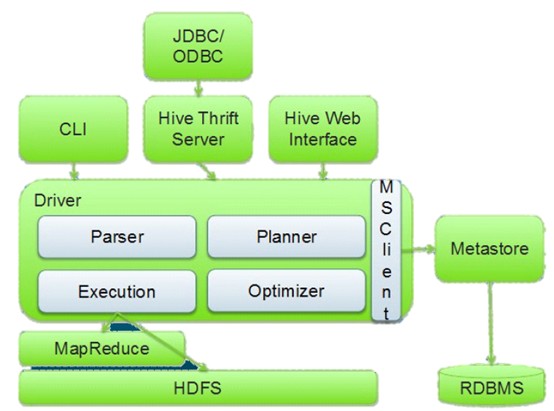


图1-1 Hive的体系结构

4Hive的数据存储

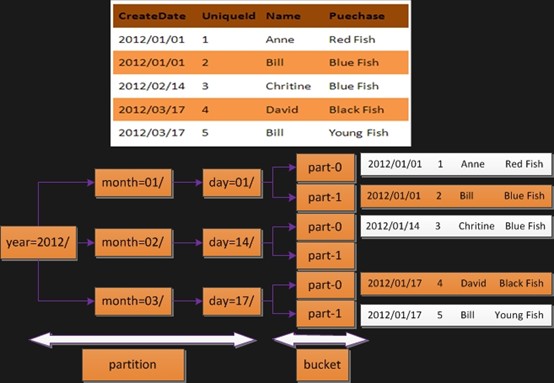
Hive的存储是建立在Hadoop文件系统之上的。Hive本身没有专门的数据存储格式，也不能为数据建立索引，用户可以自由地组织Hive中的表，只需要在创建表的时候告诉Hive数据中的列分隔符和行分隔符就可以解析数据了。

Hive中主要包含四类数据模型：表（Table）、外部表（External Table）、分区（Partition）和桶（Bucket）。

Hive中的表和数据库中的表在概念上是类似的，每个表在Hive中都有一个对应的存储目录。例如一个表pokes在HDFS中的路径为/warehouse/pokes，其中/warehouse是hive-site.xml配置文件中由${hive.metastore.warehouse.dir}指定的数据仓库的目录。

Hive中每个分区都对应数据库中相应分区列的一个索引，但是分区的组织方式和传统关系型数据库不同。在Hive中，表中的一个分区对应表下的一个目录，所有分区的数据都存储在对应的目录中。例如，图1-2中htable表中包含year、month和day三个分区，分别对应三个目录：对于year=2012，month=01，day=01的HDFS子目录为：/warehouse/htable/year=2012/ month=01/ day=01；对于year=2012，month=02，day=14的HDFS子目录为：/warehouse/htable/year=2012/ month=02/ day=14;

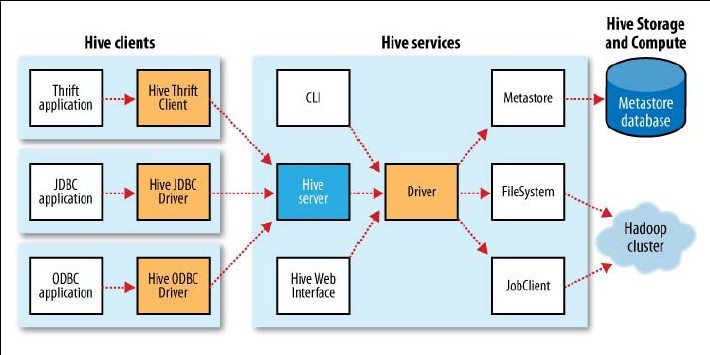
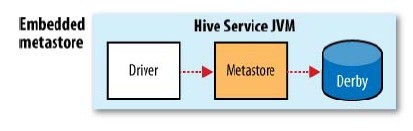
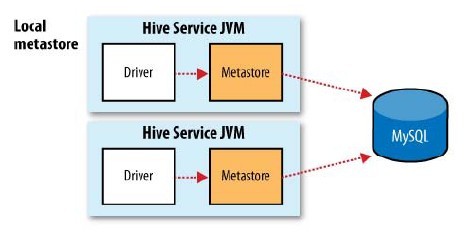
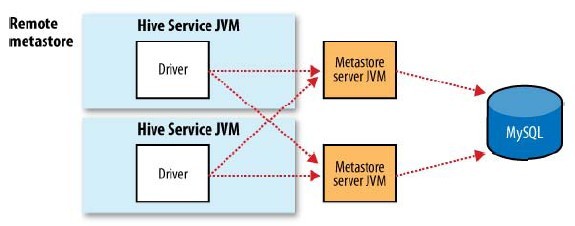
桶对**指定列**进行哈希计算时，根据哈希值切分数据，每个桶对应一个文件。例如，将图1-2中htable表中属性列Uniqueid列分散到32个桶中，首先要对Uniqueid进行hash计算，对应哈希值为0的桶写入HDFS的目录为：/warehouse/htable/year=2012/ month=01/ day=01/part-0；对应哈希值为1的桶写入HDFS的目录为：/warehouse/htable/year=2012/ month=01/ day=01/part-1。

图1-2 Hive数据存储

1. hive功能简介

功能简介PARTITIONED BY关键字为表格分区  
4.通过CLUSTERED BY关键字将PATITION划分成BUCKET  
5.定义每条记录的存储格式，包括：  
字段之间如何分隔；  
集合字段中的元素如何分隔；  
Map的key值如何分隔  
6.指定存储格式为Hadoop的SequenceFile  
  
(2)查看表结构  
DESCRIBE tablename;  
(3)修改表格  
为表格添加字段  
ALTER TABLE pokes ADD COLUMNS (new\_col INT);  
  
(4)删除表格  
DROP TABLE tablename;  
  
**DML**  
(1)、导入数据  
导入操作，只是将文件复制到对应的表格目录中，并不会对文档的schema进行校验  
从HDFS导入  
LOAD DATA INPATH 'data.txt' INTO TABLE page\_view PARTITION(date='2008-06-08', country='US')  
从本地导入，并覆盖原数据  
LOAD DATA LOCAL INPATH 'data.txt' OVERWRITE INTO TABLE page\_view PARTITION(date='2008-06-08', country='US')

Hive体系结构

  
**hiveserver**  
hiveserver启动方式：hive --service hiveserver  
HiveServer支持多种连接方式：Thrift、JDBC、ODBC  
  
**metastore**  
metastore用来存储hive的元数据信息(表格、数据库定义等)，默认情况下是和hive绑定的，部署在同一个JVM中，将元数据存储到Derby中  
这种方式不好的一点是没有办法为一个Hive开启多个实例(Derby在多个服务实例之间没有办法共享)  
  
Hive提供了增强配置，可将数据库替换成MySql等关系数据库，将存储数据独立出来在多个服务实例之间共享  
  
甚至还可以将metastore Service也独立出来，部署到其他JVM中去，在通过远程调用的方式去访问  
  
metastore的常用配置：  
hive.metastore.warehouse.dir 存储表格数据的目录  
hive.metastore.local 使用内嵌的metastore服务(默认为true)  
hive.metastore.uris 如果不使用内嵌的metastore服务，需指定远端服务的uri  
javax.jdo.option.ConnectionURL 所使用数据库的url  
javax.jdo.option.ConnectionDriverName 数据库驱动类  
javax.jdo.option.ConnectionUserName 连接用户名  
javax.jdo.option.ConnectionPassword 连接密码

hive数据存储格式

定义表格时如不指定Row Format和Stored As从句，hive采用如下默认配置：  
CREATE TABLE ...  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY '\001'  
COLLECTION ITEMS TERMINATED BY '\002'  
MAP KEYS TERMINATED BY '\003'  
LINES TERMINATED BY '\n'  
STORED AS TEXTFILE;  
默认为纯文本文件TEXTFILE  
  
如果存储的数据不是纯文本，而包含二进制的数据，可用SequenceFile和RCFile  
RCFile：基于列存储，类似于HBase，查询Table时，如果要检索的数据不是整条记录，而是具体的column，RCFile较比SequenceFile高效一些，只需遍历指定column对应的数据文件即可  
使用RCFile，创建Table时使用如下语法：  
CREATE TABLE ...  
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe'  
STORED AS RCFILE;  
  
除此之外，Hive还可通过正则表达式的方式指定输入数据源的格式：  
CREATE TABLE stations (usaf STRING, wban STRING, name STRING)  
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'  
WITH SERDEPROPERTIES (  
"input.regex" = "(\\d{6}) (\\d{5}) (.{29}) .\*"  
);  
参考资料：  
http://www.alidata.org/archives/595  
https://cwiki.apache.org/confluence/display/Hive/Home  
hadoop权威指南

1. [hive的安装和配置](http://blog.chinaunix.net/uid-451-id-3143781.html)

详细步骤1. download  
wget http://mirror.mel.bkb.net.au/pub/apache//hive/stable/hive-0.8.1.tar.gz  
tar zxf hive-0.8.1.tar.gz  
只需要在一个节点上安装  
  
  
2. 设置环境变量  
  
vi .bash\_profile  
export JAVA\_HOME=/usr/lib/jvm/java-1.6.0-openjdk-1.6.0.0.x86\_64/jre  
export HADOOP\_HOME=/home/hadoop/hadoop-1.0.0  
export HIVE\_HOME=/home/hadoop/hive-0.8.1  
export HADOOP\_CONF\_DIR=$HOME/conf  
export HIVE\_CONF\_DIR=$HOME/hive-conf  
  
export CLASSPATH=$HIVE\_HOME/lib:$JAVA\_HOME/lib:$JAVA\_HOME/jre/lib:$HADOOP\_HOME  
export PATH=$HIVE\_HOME/bin:$HADOOP\_HOME/bin:$JAVA\_HOME/bin:/sbin/:/bin:$PATH  
  
  
3. 配置hive  
cp -r hive-0.8.1/conf $HIVE\_CONF\_DIR/  
cd $HIVE\_CONF\_DIR/  
cp hive-default.xml.template hive-default.xml  
  
cat hive-env.sh  
export HADOOP\_HEAPSIZE=512  
export HIVE\_CONF\_DIR=/home/hadoop/hive-conf  
  
  
3. 测试  
$ hive  
hive> show tables;  
OK  
Time taken: 4.824 seconds  
hive> create table hwz(id int, name string);  
OK  
Time taken: 0.566 seconds  
  
hive> select \* from hwz;  
OK  
Time taken: 0.361 seconds  
  
  
  
$ hadoop dfs -lsr /user/hive  
Warning: $HADOOP\_HOME is deprecated.  
  
drwxr-xr-x - hadoop supergroup 0 2012-03-22 12:36 /user/hive/warehouse  
drwxr-xr-x - hadoop supergroup 0 2012-03-22 12:36 /user/hive/warehouse/hwz  
  
4. 配置Metastore用mysql数据库,这样才可以多用户同时访问  
a. create user and database for hive in mysql  
create database hive;  
GRANT all ON hive.\* TO hive@'%' IDENTIFIED BY 'hivepass';  
  
b. change metastore to use mysql  
cat hive-site.xml  
<configuration>  
<property>  
<name>hive.metastore.local</name>  
<value>true</value>  
</property>  
<property>  
<name>javax.jdo.option.ConnectionURL</name>  
<value>jdbc:mysql://slave1:3306/hive?createDatabaseIfNotExist=true</value>  
</property>  
<property>  
<name>javax.jdo.option.ConnectionDriverName</name>  
<value>com.mysql.jdbc.Driver</value>  
</property>  
<property>  
<name>javax.jdo.option.ConnectionUserName</name>  
<value>hive</value>  
</property>  
<property>  
<name>javax.jdo.option.ConnectionPassword</name>  
<value>hivepass</value>  
</property>  
</configuration>  
  
c. 检查  
$ hive  
hive> use dw2;  
OK

Time taken: 3.43 seconds

hive> create table hwz2(id int, name string ) row format delimited FIELDS TERMINATED BY ',';  
OK  
Time taken: 2.519 seconds

hive> show tables;

OK  
hwz2  
Time taken: 0.419 seconds  
hive> load data local inpath 'demo.txt' overwrite into table hwz2;  
Copying data from file:/home/hadoop/demo.txt  
Copying file: file:/home/hadoop/demo.txt  
Loading data to table dw2.hwz2  
Deleted hdfs://master:9000/user/hive/warehouse/dw2.db/hwz2  
OK  
Time taken: 0.557 seconds  
hive> select \* from hwz2;  
OK  
12 jack  
12 jack  
12 jack  
12 jack  
12 jack  
12 jack  
12 jack  
12 jack  
  
$ hadoop dfs -lsr /user/hive  
Warning: $HADOOP\_HOME is deprecated.  
  
drwxr-xr-x - hadoop supergroup 0 2012-03-22 15:36 /user/hive/warehouse  
drwxr-xr-x - hadoop supergroup 0 2012-03-22 15:48 /user/hive/warehouse/dw2.db  
drwxr-xr-x - hadoop supergroup 0 2012-03-22 15:48 /user/hive/warehouse/dw2.db/hwz2  
-rw-r--r-- 2 hadoop supergroup 1201 2012-03-22 15:48 /user/hive/warehouse/dw2.db/hwz2/demo.txt  
drwxr-xr-x - hadoop supergroup 0 2012-03-22 12:36 /user/hive/warehouse/hwz  
drwxr-xr-x - hadoop supergroup 0 2012-03-22 15:36 /user/hive/warehouse/hwz2  
-rw-r--r-- 2 hadoop supergroup 1201 2012-03-22 15:36 /user/hive/warehouse/hwz2/demo.txt  
$ hadoop dfs -cat /user/hive/warehouse/dw2.db/hwz2/demo.txt |head  
Warning: $HADOOP\_HOME is deprecated.  
  
12,jack  
12,jack  
12,jack  
12,jack  
12,jack  
12,jack  
12,jack  
12,jack  
12,jack  
12,jack  
  
  
  
d. 在mysql中验证建立的新表  
mysql> use hive;  
Database changed  
  
mysql> show tables;  
+-----------------+  
| Tables\_in\_hive |  
+-----------------+  
| BUCKETING\_COLS |  
| CDS |  
| COLUMNS\_V2 |  
| DATABASE\_PARAMS |  
| DBS |  
| PARTITION\_KEYS |  
| SDS |  
| SD\_PARAMS |  
| SEQUENCE\_TABLE |  
| SERDES |  
| SERDE\_PARAMS |  
| SORT\_COLS |  
| TABLE\_PARAMS |  
| TBLS |  
+-----------------+  
14 rows in set (0.00 sec)  
  
  
  
9. 常见错误  
error 1:  
-------------------------------------------------  
hive> show tables;  
FAILED: Error in metadata: javax.jdo.JDOFatalInternalException: Error creating transactional connection factory  
Solution:  
Hive不带mysql JDBC驱动,自己安装:  
  
  
wget http://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.1.18.tar.gz/from/http://mysql.mirror.kangaroot.net/  
tar zxf mysql-connector-java-5.1.18.tar.gz  
cd mysql-connector-java-5.1.18  
cp mysql-connector\*.jar $HIVE\_HOME/lib  
  
error 2:  
-------------------------------------------------  
hive> show tables;  
FAILED: Error in metadata: javax.jdo.JDOException: Couldnt obtain a new sequence (unique id) : Cannot execute statement: impossible to write to binary log since BINLOG\_FORMAT = STATEMENT and at least one table uses a storage engine limited to row-based logging. InnoDB is limited to row-logging when transaction isolation level is READ COMMITTED or READ UNCOMMITTED.  
  
Solution:  
在mysql中设置 binlog\_format='MIXED'

1. hive与hbase集成

详细步骤

一 、简介  
  
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数据库表，并提供完整的sql查询功能，可以将sql语句转换为MapReduce任务进行运行。 其优点是学习成本低，可以通过类SQL语句快速实现简单的MapReduce统计，不必开发专门的MapReduce应用，十分适合数据仓库的统计分析。  
Hive与HBase的整合功能的实现是利用两者本身对外的API接口互相进行通信，相互通信主要是依靠hive\_hbase-handler.jar工具类。  
  
二、安装步骤：  
  
1 .Hadoop和Hbase都已经成功安装了  
  
Hadoop集群配置：http://www.linuxidc.com/Linux/2012-02/53632.htm  
Hbase安装配置：http://www.linuxidc.com/Linux/2012-02/54225.htm  
  
2 . 拷贝hbase-0.90.3-cdh3u1.jar和zookeeper-3.3.3-cdh3u1.jar到hive/lib下。  
  
注意：如何hive/lib下已经存在这两个文件的其他版本（例如zookeeper-3.3.2.jar），建议删除后使用hbase下的相关版本。  
  
2. 修改hive/conf下hive-site.xml文件，在底部添加如下内容：  
  
  
<!--   
<property>  
<name>hive.exec.scratchdir</name>  
<value>/tmp</value>  
<description>Scratch space for Hive jobs</description>  
</property>  
  
<property>  
<name>hive.querylog.location</name>  
<value>/usr/local/hive/logs</value>  
</property>  
  
<property>  
<name>hive.aux.jars.path</name>  
<value>file:///usr/local/hive/lib/hive-hbase-handler-0.7.1-cdh3u1.jar,file:///usr/local/hive/lib/hbase-0.90.3-cdh3u1.jar,fi  
le:///usr/local/hive/lib/zookeeper-3.3.1.jar</value>  
</property>  
注意：如果hive-site.xml不存在则自行创建，或者把hive-default.xml.template文件改名后使用。  
  
3. 拷贝hbase-0.90.3-cdh3u1.jar到所有hadoop节点(包括master)的hadoop/lib下。  
  
4. 拷贝hbase/conf下的hbase-site.xml文件到所有hadoop节点(包括master)的hadoop/conf下。  
  
注意，如果3,4两步跳过的话，运行hive时很可能出现如下错误：  
  
  
view plaincopy   
org.apache.hadoop.hbase.ZooKeeperConnectionException: HBase is able to connect to ZooKeeper but the connection closes immediately.   
This could be a sign that the server has too many connections (30 is the default). Consider inspecting your ZK server logs for that error and   
then make sure you are reusing HBaseConfiguration as often as you can. See HTable's javadoc for more information. at org.apache.hadoop.   
hbase.zookeeper.ZooKeeperWatcher.   
  
三、启动Hive  
  
1.单节点启动  
  
#bin/hive -hiveconf hbase.dwn01=master:490001  
  
  
2 集群启动：  
  
#bin/hive -hiveconf hbase.zookeeper.quorum=dwn01,dwd01,dwd02,dwd03  
  
如何hive-site.xml文件中没有配置hive.aux.jars.path，则可以按照如下方式启动。  
  
bin/hive --auxpath /usr/local/hive/lib/hive-hbase-handler-0.8.0.jar, /usr/local/hive/lib/hbase-0.90.5.jar, /usr/local/hive/lib/zookeeper-3.3.2.jar -hiveconf hbase.zookeeper.quorum=dwn01,dwd01,dwd02,dwd03  
  
四、测试:  
  
1.创建hbase识别的数据库：  
  
  
CREATE TABLE hbase\_table\_1(key int, value string)   
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'   
WITH SERDEPROPERTIES ("hbase.columns.mapping" = ":key,cf1:val")   
TBLPROPERTIES ("hbase.table.name" = "xyz");   
hbase.table.name 定义在hbase的table名称  
  
hbase.columns.mapping 定义在hbase的列族   
2.使用sql导入数据  
  
1) 新建hive的数据表:  
  
CREATE TABLE hb\_test (id INT, url STRING);  
2)批量插入数据:  
  
hive> LOAD DATA LOCAL INPATH '/tmp/id.txt' OVERWRITE INTO TABLE hb\_test  
  
3)使用sql导入hbase\_table\_1:  
  
hive> INSERT OVERWRITE TABLE hbase\_table\_1 SELECT \* FROM hb\_test;  
  
3. 查看数据  
  
hive> select \* from hbase\_table\_1;   
  
这时可以登录Hbase去查看数据了  
#bin/hbase shell  
hbase(main):001:0> describe 'xyz'   
hbase(main):002:0> scan 'xyz'   
hbase(main):003:0> put 'xyz','100','cf1:val','www.51.com'  
  
这时在Hive中可以看到刚才在Hbase中插入的数据了。  
  
4 hive访问已经存在的hbase  
  
使用CREATE EXTERNAL TABLE:  
  
  
CREATE EXTERNAL TABLE hbase\_table\_2(key int, value string)   
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'   
WITH SERDEPROPERTIES ("hbase.columns.mapping" = "cf1:val")   
TBLPROPERTIES("hbase.table.name" = "some\_existing\_table");   
内容参考：<http://wiki.apache.org/hadoop/Hive/HBaseIntegration>

1. HIVE创建目录和表

**创建表目录**

create database companys

create database companys location table

create database companys location table with dbproperties('name'='kedde','data'='2012-01-02')

**查看database信息，无法查看当前表目录**

describe database companys

describe database extended companys

**改变默认的表目录**

use companys

**删除表目录**

drop database financials

**修改表目录属性,无法删除表属性**

alter database financials setdbproperties('owner'='dirk')

**2.表篇**

**新建表 mydb必须是存在的，location可以不存在，hive会新建**

CREATE TABLE IF NOT EXISTS mydb.employees (

name STRING COMMENT 'Employee name',

salary FLOAT COMMENT 'Employee salary',

subordinates ARRAY<STRING> COMMENT 'Names of subordinates',

deductions MAP<STRING, FLOAT>

COMMENT 'Keys are deductions names, values are percentages',

address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>

COMMENT 'Home address')

COMMENT 'Description of the table'

TBLPROPERTIES ('creator'='me', 'created\_at'='2012-01-02 10:00:00', ...)

LOCATION '/user/hive/warehouse/mydb.db/employees';

**新建表，copy已有表结构schema，不包含数据**

create table if not exists mydb.empoyees link mydb.employees;

**查看表**

show tables;

show tables in companys;

show tables 'emp.\*'; #注意是.\*

**查看表信息**

describe extended companys.tables;

**规定分隔符为/t**

row format delimited fields terminated by '/t'；

**3.表篇分区**

**不用于关系数据库partition中的字段可以不再table中，但是partition中的字段可以如同table中column一样使用这样可以加快查询速度，因为只用查找一个目下文件就可以了这里分区分为单分区partition一个column，多分区partition多个column单分区就一个目录，多分区也是一个目录，并嵌套多个目录**

实例：按照 country 和 state 给employee多分区

CREATE TABLE employees (

name STRING,

salary FLOAT,

subordinates ARRAY<STRING>,

deductions MAP<STRING, FLOAT>,

address STRUCT<street:STRING, city:STRING, state:STRING, zip:INT>

)

PARTITIONED BY (country STRING, state STRING);

**查看partition**

show partitions employees;

SHOW PARTITIONS employees PARTITION(country='US');

**添加partition(不区分大小写)**

alter table employees add partition(country='US',state='dallas')

alter table employees add partition(country='US',state='dallas') location '/home/hadoop/us-dallas'

alter table employees add partition(country='US',state='dallas') location '/home/hadoop/us-dallas' partition(country='US',state='ca') location '/home/hadoop/us-dallas'

**删除partition,分区数据和元数据都被删除**

alter table employees drop partition(country='us',state='dallas');

**向分区中添加数据**

load data inpath '/home/hadoop/resource/dallas' into table employees partition(country='us',state='dallas');

**分区的属性**

set hive.mapred.mode=strict;属性禁止没有where的语句执行在partition的table上（防止数据量巨大得table，执行这样没有限制的语句）

set hive.mapred.mode=nonstrict;

1. 数据操作

详细内容

**数据操作篇**

**加载数据（会生成partition，如果不存在的话；local字段表示为是本机目录，如果不加，则表示为hdfs上的目录；overwrite关键字表示删除目标目录，当没有则保留，但会覆盖同名旧目录）一些限制条件：使用partition，则table必须是partition的；hive不会去验证你的data是服务schema的，但是hive会验证文件是否符合表定义，如sequencefile的table，file也必须是sequence files**

load data local inpath '/home/hadoop/hive/data'

overwrite into table employees

partition(country='china',state='beijing');

**静态插入数据：**

**插入数据(select \* from employee 这样选择一个分区表，会将其partition的column也返回)**

insert overwrite table employees

partition (country='US',state='dallas')

select name,salary,subordinate,deductions,address from employees se

where se.country='china' and se.state='beijing'

**插入多条数据**

FROM staged\_employees se

INSERT OVERWRITE TABLE employees

PARTITION (country = 'US', state = 'OR')

SELECT \* WHERE se.cnty = 'US' AND se.st = 'OR'

INSERT OVERWRITE TABLE employees

PARTITION (country = 'US', state = 'CA')

SELECT \* WHERE se.cnty = 'US' AND se.st = 'CA'

INSERT OVERWRITE TABLE employees

PARTITION (country = 'US', state = 'IL')

SELECT \* WHERE se.cnty = 'US' AND se.st = 'IL';

**动态插入并创建partition：**

INSERT OVERWRITE TABLE employees

PARTITION (country, state)

SELECT ..., se.cnty, se.st

FROM staged\_employees se;

**动态静态创建partition**

INSERT OVERWRITE TABLE employees

PARTITION (country = 'US', state)

SELECT ..., se.cnty, se.st

FROM staged\_employees se

WHERE se.cnty = 'US';

**有一些参数控制，动态创建partition**

hive.exec.dynamic.partition

false

Set to true to enable dynamic partitioning.

hive.exec.dynamic.partition.mode

strict

Set to nonstrict to enable all partitions to be determined

dynamically.

hive.exec.max.dynamic.partitions.pernode

100

The maximum number of dynamic partitions that can be created

by each mapper or reducer. Raises a fatal error if one

mapper or reducer attempts to create more than the threshold.

hive.exec.max.dynamic.partitions

+1000

The total number of dynamic partitions that can be created by

one statement with dynamic partitioning. Raises a fatal error

if the limit is exceeded.

hive.exec.max.created.files

100000

The maximum total number of files that can be created globally.

A Hadoop counter is used to track the number of files created.

Raises a fatal error if the limit is exceeded.

**创建表，并加载数据**

CREATE TABLE ca\_employees

AS SELECT name, salary, address

FROM employees

WHERE se.state = 'CA';

**导出数据**

INSERT OVERWRITE LOCAL DIRECTORY '/tmp/ca\_employees'

SELECT name, salary, address

FROM employees

WHERE se.state = 'CA';

**批量导出数据**

FROM staged\_employees se

INSERT OVERWRITE DIRECTORY '/tmp/or\_employees'

SELECT \* WHERE se.cty = 'US' and se.st = 'OR'

INSERT OVERWRITE DIRECTORY '/tmp/ca\_employees'

SELECT \* WHERE se.cty = 'US' and se.st = 'CA'

INSERT OVERWRITE DIRECTORY '/tmp/il\_employees'

SELECT \* WHERE se.cty = 'US' and se.st = 'IL';

1. HIVE查询

查询的一些例子：

**1.query**

hive> SELECT name, subordinates[0] FROM employees;

John Doe Mary Smith

Mary Smith Bill King

Todd Jones NULL

**2.expression**

hive> SELECT upper(name), salary, deductions["Federal Taxes"],

round(salary \* (1 - deductions["Federal Taxes"])) FROM employees;

**3.expression**

SELECT count(\*), avg(salary) FROM employees;

**4.distinct**

SELECT count(DISTINCT symbol) FROM stocks;

**5.limit**

hive> SELECT upper(name), salary, deductions["Federal Taxes"],

> round(salary \* (1 - deductions["Federal Taxes"])) FROM employees

> LIMIT 2;

JOHN DOE 100000.0 0.2 80000

MARY SMITH 80000.0 0.2 64000

**6.列名 别名**

SELECT upper(name), salary, deductions["Federal Taxes"] as fed\_taxes,

> round(salary \* (1 - deductions["Federal Taxes"])) as

salary\_minus\_fed\_taxes

> FROM employees LIMIT 2;

**7.嵌套select （不可有having在内部select）**

hive> FROM (

> SELECT upper(name), salary, deductions["Federal Taxes"] as fed\_taxes,

> round(salary \* (1 - deductions["Federal Taxes"])) as

salary\_minus\_fed\_taxes

> FROM employees

> ) e

> SELECT e.name, e.salary\_minus\_fed\_taxes

> WHERE e.salary\_minus\_fed\_taxes > 70000;

JOHN DOE 100000.0 0.2 80000

**8. case when then**

hive> SELECT name, salary,

> CASE

> WHEN salary < 50000.0 THEN 'low'

> WHEN salary >= 50000.0 AND salary < 70000.0 THEN 'middle'

> WHEN salary >= 70000.0 AND salary < 100000.0 THEN 'high'

> ELSE 'very high'

> END AS bracket FROM employees;

John Doe 100000.0 very high

Mary Smith 80000.0 high

Todd Jones 70000.0 high

Bill King 60000.0 middle

Boss Man 200000.0 very high

**9.hive不适用map reduce**

SELECT \* FROM employees;

SELECT \* FROM employees

WHERE country = 'US' AND state = 'CA'

LIMIT 100;

**10.使用like和rlick**

like是如同sql语句

hive> SELECT name, address.street FROM employees WHERE address.street LIKE

'%Chi%';

rlick可以使用如同java的正则

hive> SELECT name, address.street

> FROM employees WHERE address.street RLIKE '.\*(Chicago|Ontario).\*';

Mary Smith 100 Ontario St.

Todd Jones 200 Chicago Ave.

**11 group by语句**

hive> SELECT year(ymd), avg(price\_close) FROM stocks

> WHERE exchange = 'NASDAQ' AND symbol = 'AAPL'

> GROUP BY year(ymd);

1984 25.578625440597534

**12 having语句**

hive> SELECT year(ymd), avg(price\_close) FROM stocks

> WHERE exchange = 'NASDAQ' AND symbol = 'AAPL'

> GROUP BY year(ymd)

> HAVING avg(price\_close) > 50.0;

1987 53.88968399108163

1991 52.49553383386182

**13 join inner（建议将最大的table放在最后）**

hive> SELECT a.ymd, a.price\_close, b.price\_close

> FROM stocks a JOIN stocks b ON a.ymd = b.ymd

> WHERE a.symbol = 'AAPL' AND b.symbol = 'IBM';

（注意，像比较这样的join条件是不允许的，条件中也不能用or）

SELECT a.ymd, a.price\_close, b.price\_close

FROM stocks a JOIN stocks b

ON a.ymd <= b.ymd

WHERE a.symbol = 'AAPL' AND b.symbol = 'IBM';

**14.left outer join**

hive> SELECT s.ymd, s.symbol, s.price\_close, d.dividend

> FROM stocks s LEFT OUTER JOIN dividends d ON s.ymd = d.ymd AND s.symbol =

**d.symbol**

> WHERE s.symbol = 'AAPL';

...

1987-05-01 AAPL 80.0 NULL

1987-05-04 AAPL 79.75 NULL

1987-05-05 AAPL 80.25 NULL

**不同类型比较**

不同类型的数字float double做比较，要注意0.2float大于0.2double

可以cat(0.2 as float)

**order by and sort by**

hive的order by是全部数据的排序，在一个reduce中处理排序，默认升序。效率比较低，通常跟limit一起用

可以用hive.mapred.mode=strict来强制，order后跟着limit

sort by是在每个reduce中，进行排序，

（是否在一个reduce中，由groupcomparator决定，如果没有就是key的compare，在hive中UDAF在控制mapper的key输出到reducer上，普通的compact是hash分布到key上，或是自定义compact的hash均匀分布ketama算法）

distribut by是让相同的key归到同一个reducer中，这样sort可以进行reducer中的排序

（注意 sort需要放在distribute前边）

**cluster**

cluster by是一种distribute和sort的简写，让按照clauses中语句分组，并且按照其他字段排序

**cast函数**

cast类型转换函数，当string不符合条件，则为unknown data为null

可以嵌套cast（cast（cast（binary as string）as double））

将float转换为int 可以用round floor

**取样查询 sample**

rand随机

select \* from numbers tablesample(bucket 3 out of 10 on rand()) s;

column随机：以一个column，这样会在多个线程里跑multiple runs

select \* from numbers tablesample(bucket 3 out of 10 on number) s;

block随机: 另一个取样函数(block sampling 当表小于block size128mb，则全部rows返回)

有个hive.sample.seednumber来控制seed information for block based

select \* from numbersflat tablesample(0.1 percent) s;

使用number做sample,用以下语句，让sample按照sample存储，可以只访问hash file中需要的bucket

create table numbers\_bucketed(number int) clustered by (number) into 3 buckets

set hive.enforce.bucketing = true;

**union**

select \* from table1

union all

select \* from table2

from(

from src select src.key , src.value where src.key < 100

union all

from src select src.\* where src.key >100

) unioninput

insert overwrite directory '/tmp/union.out' select unioninput.\*

1. HIVE视图

**view1**

create view shorter\_join as

select \* from people join cart

on(cart.people\_id=people.id) where firstname='dirk'

select lastname from shorter\_join where id=3

**view2**

create view if not exists shipments(time,part)

comment 'time and parts for shipments'

tblproperties('creator'='me')

as select...;

**通过试图建table**

create table shipments2 like shipments

**删除view**

drop view if exists shipments

1. 索引

**索引**

**创建索引**

create index employees\_index

on table employees(country)

as 'bitmap' #使用bitmap函数建立索引

with deferred rebuild

idxproperties('creator'='dirk','created\_at'='some\_time')

in table employees\_index\_table

partitioned by (country,name)

comment 'employees indexed by country and name'

**显示表的索引 #formatted是可选**

show formatted index on employees

show formatted indexed on employees

**删除索引表（不确定）**

drop index if exists employees\_index on table employees;

**索引表的handler，会初始化，定义，验证索引表的schema**

在索引建立时，会使用rebuilding process，读取表为索引表并且写到索引数据库中，所有的存储都会

索引删除时，会删除明显的索引数据库

handler会参与优化查询

1. hive schema

**schema设计**

hive pattern && hive anti-pattern

**1.Table by day 按照天分割数据，在relation中，这个参数不推荐，在hive中使用**

create table supply(id int,part string,quantity int) partitioned by (int day)

alter table supply add partition (day=20120102)

partition的负面影响:

1.namenode limition

但是partition产生的子目录，子文件都会保存在hdfs中，namenode会存在内存中，所以这得负面效果是namenode的filesystem的容量上限(hadoop has this upper limit on the total number of file,mapr and amazon s3 don't have this limitation)

2.一个job分解成几个task，每个task是一个jvm实例，每一个file对应一个独立的task，每个task是jvm中独立的一个实例（进程），过多的实例会给jvm压力（start up and tear down），这使得计算速度降低

因此不能有太多partition，每个文件要尽可能的大

一个好的table by day的设计，是设计出相似大小的数据在不同的时间间断，时间间断可以适当增大。同时保证每个file大于filesystem block size。目的是让partition足够的大。另一种方法，是用多维度的partition分解数据。

**2.unique keys and normalization 主键，格式化数据**

关系数据库最爱用地策略，但是在hive中没有这种概念。因为hive可以存储denormalized data非格式化的数据，如array,map,struct。这样可以避免one-to-many的关联关系，加快了io速度。但是也pay the penalty of denormalization，比如数据复制，数据不一致的概率

**3.making multiple passes over the same data 同数据源的操作优化**

insert overwrite table sales

select \* from history where action='purchased';

insert overwrite table credits

select \* from history where action='returned';

from history

insert overwrite sales select \* where action='purchased'

insert overwrite credits select \* where action = 'returned'

**4.the case for partitioning every table**

为了避免job fail而使得数据被删除，在insert数据的时候可以使用table pardae table1 partition(day=20120102).但是需要删除这个中间换转者partition

**5.bucketing table data storage**

当table没有明显的partition特征时，或是减轻filesystem的负担,可以使用bucketing,他的优点是不会随着增加数据使得文件个数变动，而且对于取样sample是很容易的，对于一些joins操作也比较便利。

create table weblog(user\_id int,url string,source\_ip string) partition by (dt string) clustered by (user\_id) into 96 buckets;

为了生成正确个数的reducer对应hash出得bucket

在查询的时候设置 set hive.enforce.bucketing=true;

from raw\_logs或是设置reduce数直接等于bucket数set mapred.reduce.tasks=96

insert overwrite table weblog partition(dt='2009-02-25') select user\_id,url,source\_ip where dt='2009-02-25'

**6.adding colums to a table**

hive是没有格式化的数据仓库，随着数据需求可以增加一列，数据少于期待列数，则填补null，数据多于，则舍弃。

create table weblogs(version long,url string) partitioned by (hit\_data int) row format delimited fields terminated by '\t'

加载数据，可以用int补上缺少的数据

load data local inpath 'log1.txt' int weblogs partition(20110101)

**7.(almost)always use compression**

1. Hive join

Join的语法规则：  
join\_table:  
table\_reference JOIN table\_factor [join\_condition]  
| table\_reference {LEFT|RIGHT|FULL} [OUTER]  
JOIN table\_reference join\_condition  
| table\_reference LEFT SEMI JOIN  
table\_reference join\_condition   
  
table\_reference:  
table\_factor  
| join\_table  
  
table\_factor:  
tbl\_name [alias]  
| table\_subquery alias  
| ( table\_references )  
  
join\_condition:  
ON equality\_expression ( AND equality\_expression )\*  
  
equality\_expression:  
expression = expression   
Hive 只支持等值连接（equality joins）、外连接（outer joins）和（left semi joins）。  
Hive 不支持所有非等值的连接，因为非等值连接非常难转化到 map/reduce 任务。另外，Hive 支持多于 2 个表的连接。  
  
join left/right outer join:一定输出左边/右边的每一行对应结果（其实都可以转换为做连接）  
left semi join 用于实现a.key in select key from table b（即in/exist功能）  
1. 只支持等值join，例如：  
  
SELECT a.\* FROM a JOIN b ON (a.id = b.id)  
SELECT a.\* FROM a JOIN b  
ON (a.id = b.id AND a.department = b.department)  
是正确的，然而:  
SELECT a.\* FROM a JOIN b ON (a.id b.id)  
是错误的。   
  
2. 可以 join 多于 2 个表，例如  
  
SELECT a.val, b.val, c.val FROM a JOIN b  
ON (a.key = b.key1) JOIN c ON (c.key = b.key2)  
如果join中多个表的 join key 是同一个，则 join 会被转化为单个 map/reduce 任务，例如：  
SELECT a.val, b.val, c.val FROM a JOIN b  
ON (a.key = b.key1) JOIN c  
ON (c.key = b.key1)  
被转化为单个 map/reduce 任务，因为 join 中只使用了 b.key1 作为 join key。  
  
SELECT a.val, b.val, c.val FROM a JOIN b ON (a.key = b.key1)  
JOIN c ON (c.key = b.key2)  
而这一 join 被转化为 2 个 map/reduce 任务。因为 b.key1 用于第一次 join 条件，而 b.key2 用于第二次 join。  
  
join 时，每次 map/reduce 任务的逻辑是这样的：reducer 会缓存 join 序列中除了最后一个表的所有表的记录，再通过最后一个表将结果序列化到文件系统。  
这一实现有助于在 reduce 端减少内存的使用量。实践中，应该把最大的那个表写在最后（否则会因为缓存浪费大量内存）。例如：  
  
SELECT a.val, b.val, c.val FROM a  
JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key1)  
  
所有表都使用同一个 join key（使用 1 次 map/reduce 任务计算）。Reduce 端会缓存 a 表和 b 表的记录，然后每次取得一个 c 表的记录就计算一次 join 结果，类似的还有：  
  
SELECT a.val, b.val, c.val FROM a  
JOIN b ON (a.key = b.key1) JOIN c ON (c.key = b.key2)  
  
这里用了 2 次 map/reduce 任务。第一次缓存 a 表，用 b 表序列化；第二次缓存第一次 map/reduce 任务的结果，然后用 c 表序列化。   
  
LEFT，RIGHT 和 FULL OUTER 关键字用于处理 join 中空记录的情况，例如：  
  
SELECT a.val, b.val FROM a LEFT OUTER  
JOIN b ON (a.key=b.key)  
  
对应所有 a 表中的记录都有一条记录输出。当 a.key=b.key 时,输出的结果应该是 a.val, b.val;而当 b.key 中找不到等值的 a.key 记录时也会输出 a.val, NULL。  
“FROM a LEFT OUTER JOIN b”这句一定要写在同一行——意思是 a 表在 b 表的左边，所以 a 表中的所有记录都被保留了；“a RIGHT OUTER JOIN b”会保留所有 b 表的记录。OUTER JOIN 语义应该是遵循标准 SQL spec的。  
  
Join 发生在 WHERE 子句之前。  
如果你想限制 join 的输出，应该在 WHERE 子句中写过滤条件——或是在 join 子句中写。这里面一个容易混淆的问题是表分区的情况：  
  
SELECT a.val, b.val FROM a  
LEFT OUTER JOIN b ON (a.key=b.key)  
WHERE a.ds='2009-07-07' AND b.ds='2009-07-07'  
  
会 join a 表到 b 表（OUTER JOIN），列出 a.val 和 b.val 的记录。WHERE 从句中可以使用其他列作为过滤条件。但是，如前所述，如果 b 表中找不到对应 a 表的记录，b 表的所有列都会列出 NULL，包括 ds 列。也就是说，join 会过滤 b 表中不能找到匹配 a 表 join key 的所有记录。这样的话，LEFT OUTER 就使得查询结果与 WHERE 子句无关了。解决的办法是在 OUTER JOIN 时使用以下语法：  
  
SELECT a.val, b.val FROM a LEFT OUTER JOIN b  
ON (a.key=b.key AND  
b.ds='2009-07-07' AND  
a.ds='2009-07-07')  
  
这一查询的结果是预先在 join 阶段过滤过的，所以不会存在上述问题。这一逻辑也可以应用于 RIGHT 和 FULL 类型的 join 中。  
  
Join 是不能交换位置的。无论是 LEFT 还是 RIGHT join，都是左连接的。  
  
SELECT a.val1, a.val2, b.val, c.val  
FROM a  
JOIN b ON (a.key = b.key)  
LEFT OUTER JOIN c ON (a.key = c.key)  
  
先 join a 表到 b 表，丢弃掉所有 join key 中不匹配的记录，然后用这一中间结果和 c 表做 join。这一表述有一个不太明显的问题，就是当一个 key 在 a 表和 c 表都存在，但是 b 表中不存在的时候：整个记录在第一次 join，即 a JOIN b 的时候都被丢掉了（包括a.val1，a.val2和a.key），然后我们再和 c 表 join 的时候，如果 c.key 与 a.key 或 b.key 相等，就会得到这样的结果：NULL, NULL, NULL, c.val。   
  
LEFT SEMI JOIN 是 IN/EXISTS 子查询的一种更高效的实现。  
Hive 当前没有实现 IN/EXISTS 子查询，所以你可以用 LEFT SEMI JOIN 重写你的子查询语句。LEFT SEMI JOIN 的限制是， JOIN 子句中右边的表只能在 ON 子句中设置过滤条件，在 WHERE 子句、SELECT 子句或其他地方过滤都不行。  
  
SELECT a.key, a.value  
FROM a  
WHERE a.key in  
(SELECT b.key  
FROM B);  
可以被重写为：  
SELECT a.key, a.val  
FROM a LEFT SEMI JOIN b on (a.key = b.key)

1. Hive基本语法

DDL Operations   
创建表   
hive> CREATE TABLE pokes (foo INT, bar STRING);   
创建表并创建索引字段ds   
hive> CREATE TABLE invites (foo INT, bar STRING) PARTITIONED BY (ds STRING);   
显示所有表   
hive> SHOW TABLES;   
按正条件（正则表达式）显示表，   
hive> SHOW TABLES '.\*s';   
表添加一列   
hive> ALTER TABLE pokes ADD COLUMNS (new\_col INT);   
添加一列并增加列字段注释   
hive> ALTER TABLE invites ADD COLUMNS (new\_col2 INT COMMENT 'a comment');   
更改表名   
hive> ALTER TABLE events RENAME TO 3koobecaf;   
删除列   
hive> DROP TABLE pokes;   
元数据存储   
将文件中的数据加载到表中   
hive> LOAD DATA LOCAL INPATH './examples/files/kv1.txt' OVERWRITE INTO TABLE pokes;   
加载本地数据，同时给定分区信息   
hive> LOAD DATA LOCAL INPATH './examples/files/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');   
加载DFS数据 ，同时给定分区信息   
hive> LOAD DATA INPATH '/user/myname/kv2.txt' OVERWRITE INTO TABLE invites PARTITION (ds='2008-08-15');   
The above command will load data from an HDFS file/directory to the table. Note that loading data from HDFS will result in moving the file/directory. As a result, the operation is almost instantaneous.   
SQL 操作   
按先件查询   
hive> SELECT a.foo FROM invites a WHERE a.ds='<DATE>';   
将查询数据输出至目录   
hive> INSERT OVERWRITE DIRECTORY '/tmp/hdfs\_out' SELECT a.\* FROM invites a WHERE a.ds='<DATE>';   
将查询结果输出至本地目录   
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/local\_out' SELECT a.\* FROM pokes a;   
选择所有列到本地目录   
hive> INSERT OVERWRITE TABLE events SELECT a.\* FROM profiles a;   
hive> INSERT OVERWRITE TABLE events SELECT a.\* FROM profiles a WHERE a.key < 100;   
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/reg\_3' SELECT a.\* FROM events a;   
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg\_4' select a.invites, a.pokes FROM profiles a;   
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg\_5' SELECT COUNT(1) FROM invites a WHERE a.ds='<DATE>';   
hive> INSERT OVERWRITE DIRECTORY '/tmp/reg\_5' SELECT a.foo, a.bar FROM invites a;   
hive> INSERT OVERWRITE LOCAL DIRECTORY '/tmp/sum' SELECT SUM(a.pc) FROM pc1 a;   
将一个表的统计结果插入另一个表中   
hive> FROM invites a INSERT OVERWRITE TABLE events SELECT a.bar, count(1) WHERE a.foo > 0 GROUP BY a.bar;   
hive> INSERT OVERWRITE TABLE events SELECT a.bar, count(1) FROM invites a WHERE a.foo > 0 GROUP BY a.bar;   
JOIN   
hive> FROM pokes t1 JOIN invites t2 ON (t1.bar = t2.bar) INSERT OVERWRITE TABLE events SELECT t1.bar, t1.foo, t2.foo;   
将多表数据插入到同一表中   
FROM src   
INSERT OVERWRITE TABLE dest1 SELECT src.\* WHERE src.key < 100   
INSERT OVERWRITE TABLE dest2 SELECT src.key, src.value WHERE src.key >= 100 and src.key < 200   
INSERT OVERWRITE TABLE dest3 PARTITION(ds='2008-04-08', hr='12') SELECT src.key WHERE src.key >= 200 and src.key < 300   
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/dest4.out' SELECT src.value WHERE src.key >= 300;   
将文件流直接插入文件   
hive> FROM invites a INSERT OVERWRITE TABLE events SELECT TRANSFORM(a.foo, a.bar) AS (oof, rab) USING '/bin/cat' WHERE a.ds > '2008-08-09';   
This streams the data in the map phase through the script /bin/cat (like hadoop streaming). Similarly - streaming can be used on the reduce side (please see the Hive Tutorial or examples)   
实际示例   
创建一个表   
CREATE TABLE u\_data (   
userid INT,   
movieid INT,   
rating INT,   
unixtime STRING)   
ROW FORMAT DELIMITED   
FIELDS TERMINATED BY '\t'   
STORED AS TEXTFILE;   
下载示例数据文件，并解压缩   
wget http://www.grouplens.org/system/files/ml-data.tar\_\_0.gz   
tar xvzf ml-data.tar\_\_0.gz   
加载数据到表中   
LOAD DATA LOCAL INPATH 'ml-data/u.data'   
OVERWRITE INTO TABLE u\_data;   
统计数据总量   
SELECT COUNT(1) FROM u\_data;   
现在做一些复杂的数据分析   
创建一个 weekday\_mapper.py: 文件，作为数据按周进行分割   
import sys   
import datetime   
for line in sys.stdin:   
line = line.strip()   
userid, movieid, rating, unixtime = line.split('\t')   
生成数据的周信息   
weekday = datetime.datetime.fromtimestamp(float(unixtime)).isoweekday()   
print '\t'.join([userid, movieid, rating, str(weekday)])   
使用映射脚本   
//创建表，按分割符分割行中的字段值   
CREATE TABLE u\_data\_new (   
userid INT,   
movieid INT,   
rating INT,   
weekday INT)   
ROW FORMAT DELIMITED   
FIELDS TERMINATED BY '\t';   
//将python文件加载到系统   
add FILE weekday\_mapper.py;   
将数据按周进行分割   
INSERT OVERWRITE TABLE u\_data\_new   
SELECT   
TRANSFORM (userid, movieid, rating, unixtime)   
USING 'python weekday\_mapper.py'   
AS (userid, movieid, rating, weekday)   
FROM u\_data;   
SELECT weekday, COUNT(1)   
FROM u\_data\_new   
GROUP BY weekday;   
处理Apache Weblog 数据   
将WEB日志先用正则表达式进行组合，再按需要的条件进行组合输入到表中   
add jar ../build/contrib/hive\_contrib.jar;   
CREATE TABLE apachelog (   
host STRING,   
identity STRING,   
user STRING,   
time STRING,   
request STRING,   
status STRING,   
size STRING,   
referer STRING,   
agent STRING)   
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'   
WITH SERDEPROPERTIES (   
"input.regex" = "([^ ]\*) ([^ ]\*) ([^ ]\*) (-|\\[[^\\]]\*\\]) ([^ \"]\*|\"[^\"]\*\") (-|[0-9]\*) (-|[0-9]\*)(?: ([^ \"]\*|\"[^\"]\*\") ([^ \"]\*|\"[^\"]\*\"))?",   
"output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s"   
)   
STORED AS TEXTFILE;

1. Hive操作语句
2. 创建表的语句：  
   Create [EXTERNAL] TABLE [IF NOT EXISTS] table\_name   
   [(col\_name data\_type [COMMENT col\_comment], ...)]   
   [COMMENT table\_comment]   
   [PARTITIONED BY (col\_name data\_type [COMMENT col\_comment], ...)]   
   [CLUSTERED BY (col\_name, col\_name, ...) [SORTED BY (col\_name [ASC|DESC], ...)]INTO num\_buckets BUCKETS]  
   [ROW FORMAT row\_format]   
   [STORED AS file\_format]   
   [LOCATION hdfs\_path]  
     
   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 EXIST 选项来忽略这个异常。  
   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。  
   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 SEQUENCE 。  
   有分区的表可以在创建的时候使用 PARTITIONED BY 语句。一个表可以拥有一个或者多个分区，每一个分区单独存在一个目录下。而且，表和分区都可以对某个列进行 CLUSTERED BY 操作，将若干个列放入一个桶（bucket）中。也可以利用SORT BY 对数据进行排序。这样可以为特定应用提高性能。  
     
     
     
   创建普通的表：  
   create table test\_table (  
     
   id int,  
     
   name string,  
     
   no int)   
     
   row formatdelimited   
     
   fieldsterminated by ','   
     
   stored astextfile;//指定了字段的分隔符，hive只支持单个字符的分隔符。hive默认的分隔符是\001  
     
     
     
   创建带有partition的表：  
   create table test\_partition (  
     
   id int,  
     
   name string,  
     
   no int)  
     
   partitioned by(dt string)   
     
   row formatdelimited   
     
   fieldsterminated by ','   
     
   stored astextfile;  
     
     
   load data local inpath '/home/zhangxin/hive/test\_hive.txt' overwrite into tabletest\_partition partition (dt='2012-03-05');  
     
     
     
   创建带有Bucket的表：  
   create table test\_bucket (  
     
   id int,  
     
   name string,  
     
   no int)  
     
   partitioned by(dt string)   
     
   clustered by(id) into 10 buckets --将id这一列分到10个桶中。  
     
   row formatdelimited   
     
   fieldsterminated by ','   
     
   stored astextfile;  
     
   关于分桶：对列进行分桶，本质上，在进行reduce的时候，会对列的值进行hash，然后将hash的值放到特定的桶中（有点像distribute by）。  
     
   注意在写入分桶数据的时候，需要指定：  
     
   sethive.enforce.bucketing=true;  
     
   更多关于bucketed table的信息，见：  
     
   http://hive.apache.org/docs/r0.8.1/language\_manual/working\_with\_bucketed\_tables.html  
     
     
   创建external表：  
   create external table test\_external (  
     
   id int,  
     
   name string,  
     
   no int)  
     
   row formatdelimited   
     
   fieldsterminated by ','   
     
   location'/home/zhangxin/hive/test\_hive.txt';  
     
     
   创建与已知表相同结构的表Like：  
     
   只复制表的结构，而不复制表的内容。  
   create table test\_like\_table like test\_bucket;  
     
   2.删除表：  
   Drop Table tablename；  
     
   删除一个内部表的同时会同时删除表的元数据和数据。删除一个外部表，只删除元数据而保留数据。  
     
   3.修改已经存在的表：  
   alter table  
   Alter table 语句允许用户改变现有表的结构。用户可以增加列/分区,表本身重命名。  
     
   1) 增加分区 Add Partitions：  
   ALTER TABLE table\_name ADD partition\_spec [ LOCATION 'location1' ]partition\_spec [ LOCATION 'location2' ]  
   其中partition\_spec 的格式为：PARTITION (partition\_col =partition\_col\_value, partition\_col = partiton\_col\_value, ...)  
   用户可以用 ALTER TABLE ADD PARTITION 来向一个表中增加分区。当分区名是字符串时加引号。  
     
   alter table test\_partition add partition (dt='2012-03-06') location'/home/zhangxin/hive/test\_hive.txt';  
     
   2) 删除分区 drop Partition：  
   ALTER TABLE table\_name DROP partition\_spec, partition\_spec,...  
   用户可以用 ALTER TABLE DROP PARTITION 来删除分区。分区的元数据和数据将被一并删除。  
     
   alter table test\_partition drop partition (dt='2012-03-06')  
     
   3) 对表进行重命名 rename to：  
   ALTER TABLE table\_name RENAME TO new\_table\_name  
   这个命令可以让用户为表更名。数据所在的位置和分区名并不改变。换而言之，老的表名并未“释放”，对老表的更改会改变新表的数据。  
   alter table test\_partition rename to new\_test\_partition;  
     
   4) 对表中的某一列进行修改，包括列的名称/列的数据类型/列的位置/列的注释  
   ALTER TABLE table\_name CHANGE [COLUMN] col\_old\_name col\_new\_name column\_type[COMMENT col\_comment] [FIRST|AFTER column\_name]  
   这个命令可以允许用户修改一个列的名称、数据类型、注释或者位置  
   create table test\_col\_change (a int,b int, c int);  
     
   修改列的名称，后面一定要加上数据类型：  
   ALTER TABLE test\_col\_change CHANGE a a1 INT; 将 a 列的名字改为 a1.  
     
   ALTER TABLE test\_col\_change CHANGE a a1 STRING AFTER b; 将 a 列的名字改为 a1，a 列的数据类型改为 string，并将它放置在列 b 之后。新的表结构为： b int, a1 string, c int.  
     
   ALTER TABLE test\_col\_change CHANGE b b1 INT FIRST; 会将 b 列的名字修改为 b1, 并将它放在第一列。新表的结构为： b1 int, a string, c int.  
     
   注意：对列的改变只会修改Hive 的元数据，而不会改变实际数据。用户应该确定保证元数据定义和实际数据结构的一致性。  
     
   5) 添加/替换列Add/ReplaceColumns  
   ALTER TABLE table\_name ADD|REPLACE COLUMNS (col\_name data\_type [COMMENTcol\_comment], ...)  
   ADD COLUMNS 允许用户在当前列的末尾增加新的列，但是在分区列之前。  
   alter table test\_col\_change add columns (d int);  
   describe test\_col\_change;  
   OK  
   a1 int   
   b1 string   
   c int   
   d int   
     
   REPLACE COLUMNS 删除以后的列，加入新的列。只有在使用 native 的 SerDE（DynamicSerDe orMetadataTypeColumnsetSerDe）的时候才可以这么做。  
   alter table test\_col\_change replace columns (c int);  
   describetest\_col\_change;   
   OK  
   c int   
     
   6) 修改表的属性Alter Table Properties:  
   ALTER TABLE table\_name SET TBLPROPERTIES table\_properties   
   table\_properties: : (property\_name = property\_value, property\_name =property\_value, ... )  
   用 户可以用这个命令向表中增加 metadata，目前 last\_modified\_user，last\_modified\_time 属性都是由 Hive 自动管理的。用户可以向列表中增加自己的属性。可以使用 DESCRIBE EXTENDED TABLE 来获得这些信息。  
   alter table test\_col\_change set tblproperties ('key1'='value1');  
   可以通过 describe extended test\_col\_change; 查看表的属性信息。  
     
   7) 修改表的序列化和反序列化属性：  
   ALTER TABLE table\_name SET SERDE serde\_class\_name [WITH SERDEPROPERTIESserde\_properties]  
   ALTER TABLE table\_name SET SERDEPROPERTIES serde\_properties  
   serde\_properties: : (property\_name = property\_value, property\_name =property\_value, ... )  
     
   这个命令允许用户向 SerDe 对象增加用户定义的元数据。Hive 为了序列化和反序列化数据，将会初始化 SerDe 属性，并将属性传给表的 SerDe。如此，用户可以为自定义的 SerDe 存储属性。  
     
   8) 修改表的文件存储格式组织方式：  
   ALTER TABLE table\_name SET FILEFORMAT file\_format  
   ALTER TABLE table\_name CLUSTERED BY (col\_name, col\_name, ...) [SORTED BY(col\_name, ...)] INTO num\_buckets BUCKETS  
   这个命令修改了表的物理存储属性。  
     
   4.将数据加载到表中：  
   LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION(partcol1=val1, partcol2=val2 ...)]  
   当数据被加载至表中时，不会对数据进行任何转换。Load 操作只是将数据复制/移动至 Hive 表对应的位置。  
   filepath 可以是：  
   相对路径，例如：project/data1  
   绝对路径，例如： /user/hive/project/data1  
   包含模式的完整 URI，例如：hdfs://namenode:9000/user/hive/project/data1  
   加载的目标可以是一个表或者分区。如果表包含分区，必须指定每一个分区的分区名。  
   filepath 可以引用一个文件（这种情况下，Hive 会将文件移动到表所对应的目录中）或者是一个目录（在这种情况下，Hive 会将目录中的所有文件移动至表所对应的目录中）。  
   如 果指定了 LOCAL，那么：load 命令会去查找本地文件系统中的 filepath。如果发现是相对路径，则路径会被解释为相对于当前用户的当前路径。用户也可以为本地文件指定一个完整的 URI，比如：file:///user/hive/project/data1.load 命令会将 filepath 中的文件复制到目标文件系统中。  
   目标文件系统由表的位置属性决定。被复制的数据文件移动到表的数据对应的位置。如果没有指定 LOCAL 关键字，如果 filepath 指向的是一个完整的 URI，hive 会直接使用这个 URI。 否则：如果没有指定 schema 或者 authority，Hive 会使用在 hadoop 配置文件中定义的 schema 和 authority，fs.default.name 指定了 Namenode 的 URI。如果路径不是绝对的，Hive 相对于 /user/ 进行解释。Hive 会将 filepath 中指定的文件内容移动到 table （或者 partition）所指定的路径中。如果使用了 OVERWRITE 关键字，则目标表（或者分区）中的内容（如果有）会被删除，然后再将 filepath 指向的文件/目录中的内容添加到表/分区中。如果目标表（分区）已经有一个文件，并且文件名和 filepath 中的文件名冲突，那么现有的文件会被新文件所替代。

Hive shell命令

一、Hive命令选项  
Usage:

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. Usage: hive [-hiveconf x=y]\* [<-i filename>]\* [<-f filename>|<-e query-string>] [-S]
2. -i <filename> Initialization Sql from file (executed automatically and silently before any other commands)
3. -e 'quoted query string' Sql from command line
4. -f <filename> Sql from file
5. -S Silent mode in interactive shell where only data is emitted
6. -hiveconf x=y Use this to set hive/hadoop configuration variables.
7. -e and -f cannot be specified together. In the absence of these options, interactive shell is started. However, -i can be used with any other options.
8. To see this usage help, run hive -h

Usage: hive [-hiveconf x=y]\* [<-i filename>]\* [<-f filename>|<-e query-string>] [-S]

-i <filename> Initialization Sql from file (executed automatically and silently before any other commands)

-e 'quoted query string' Sql from command line

-f <filename> Sql from file

-S Silent mode in interactive shell where only data is emitted

-hiveconf x=y Use this to set hive/hadoop configuration variables.

-e and -f cannot be specified together. In the absence of these options, interactive shell is started. However, -i can be used with any other options.

To see this usage help, run hive -h

下面的例子是做一个命令行的查询：

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. $HIVE\_HOME/bin/hive -e 'select a.col from tab1 a'

$HIVE\_HOME/bin/hive -e 'select a.col from tab1 a'

下面的例子是指定Hive配置查询：

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. $HIVE\_HOME/bin/hive -e 'select a.col from tab1 a' -hiveconf hive.exec.scratchdir=/home/my/hive\_scratch -hiveconf mapred.reduce.tasks=32

$HIVE\_HOME/bin/hive -e 'select a.col from tab1 a' -hiveconf hive.exec.scratchdir=/home/my/hive\_scratch -hiveconf mapred.reduce.tasks=32

下面的例子是将查询结果导入到文本文件：

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. $HIVE\_HOME/bin/hive -S -e 'select a.col from tab1 a' > a.txt

$HIVE\_HOME/bin/hive -S -e 'select a.col from tab1 a' > a.txt

下面的例子是使用SQL文件进行操作：

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. $HIVE\_HOME/bin/hive -f /home/my/hive-script.sql

$HIVE\_HOME/bin/hive -f /home/my/hive-script.sql

下面的例子是在进入交互式界面之前跑一个初始化的脚本：

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. $HIVE\_HOME/bin/hive -i /home/my/hive-init.sql

$HIVE\_HOME/bin/hive -i /home/my/hive-init.sql

二、hiverc file  
如果没有-i参数，那么hive会直接进入命令行界面，同时会加载HIVE\_HOME/bin/.hiverc and $HOME/.hiverc作为初始化所需要的文件  
  
  
三、hive交互的Shell命令

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. Command Description
2. quit Use quit or exit to leave the interactive shell.
3. set key=value Use this to set value of particular configuration variable. One thing to note here is that if you misspell the variable name, cli will not show an error.
4. set This will print a list of configuration variables that are overridden by user or hive.
5. set -v This will print all hadoop and hive configuration variables.
6. add FILE [file] [file]\* Adds a file to the list of resources
7. list FILE list all the files added to the distributed cache
8. list FILE [file]\* Check if given resources are already added to distributed cache
9. ! [cmd] Executes a shell command from the hive shell
10. dfs [dfs cmd] Executes a dfs command from the hive shell
11. [query] Executes a hive query and prints results to standard out
12. source FILE Used to execute a script file inside the CLI.

Command Description

quit Use quit or exit to leave the interactive shell.

set key=value Use this to set value of particular configuration variable. One thing to note here is that if you misspell the variable name, cli will not show an error.

set This will print a list of configuration variables that are overridden by user or hive.

set -v This will print all hadoop and hive configuration variables.

add FILE [file] [file]\* Adds a file to the list of resources

list FILE list all the files added to the distributed cache

list FILE [file]\* Check if given resources are already added to distributed cache

! [cmd] Executes a shell command from the hive shell

dfs [dfs cmd] Executes a dfs command from the hive shell

[query] Executes a hive query and prints results to standard out

source FILE Used to execute a script file inside the CLI.

例子：

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. hive> set mapred.reduce.tasks=32;
2. hive> set;
3. hive> select a.\* from tab1;
4. hive> !ls;
5. hive> dfs -ls;

hive> set mapred.reduce.tasks=32;

hive> set;

hive> select a.\* from tab1;

hive> !ls;

hive> dfs -ls;

四、Hive日志  
  
Hive使用Log4j写日志，这些日志将不会以标准输出方式进行输出，默认情况Hive将使用hive-log4j，配置文件在conf目录下，日志输出在 /tmp/$USER/hive.log 下，日志级别为WARN。  
  
为了Debug，你可以修改日志的输出格式以及改变日志的输出级别，你可以在命令行下使用以下命令：

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. $HIVE\_HOME/bin/hive -hiveconf hive.root.logger=INFO,console

$HIVE\_HOME/bin/hive -hiveconf hive.root.logger=INFO,console

hive.root.logger 指定了日志的级别以及日志输出位置，输出在控制台。这样日志不会输出到文件中。  
  
五、Hive 资源  
hive可以管理查询有效的附加资源到Session中。任何本地的acessible文件会加入到这个session，hive加载这个文件到session中后可以进行相关的map/reduce任务，hive使用haddop cache来处理被加载的文件。

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. ADD { FILE[S] | JAR[S] | ARCHIVE[S] } <filepath1> [<filepath2>]\*
2. LIST { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]
3. DELETE { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]

ADD { FILE[S] | JAR[S] | ARCHIVE[S] } <filepath1> [<filepath2>]\*

LIST { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]

DELETE { FILE[S] | JAR[S] | ARCHIVE[S] } [<filepath1> <filepath2> ..]

文件资源仅被添加到目标cache中。Jar资源将被添加到Java classpath中。ARCHIVE资源将被自动添加来描述他们。  
例如：

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. hive> add FILE /tmp/tt.py;
2. hive> list FILES;
3. /tmp/tt.py
4. hive> from networks a MAP a.networkid USING 'python tt.py' as nn where a.ds = '2009-01-04' limit 10;

hive> add FILE /tmp/tt.py;

hive> list FILES;

/tmp/tt.py

hive> from networks a MAP a.networkid USING 'python tt.py' as nn where a.ds = '2009-01-04' limit 10;

如果命令在所有节点上均有效就没有必要加入到Session中. For example:

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704255)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. ... MAP a.networkid USING 'wc -l' ...: here wc is an executable available on all machines
2. ... MAP a.networkid USING '/home/nfsserv1/hadoopscripts/tt.py' ...: here tt.py may be accessible via a nfs mount point that's configured identically on all the

... MAP a.networkid USING 'wc -l' ...: here wc is an executable available on all machines

... MAP a.networkid USING '/home/nfsserv1/hadoopscripts/tt.py' ...: here tt.py may be accessible via a nfs mount point that's configured identically on all the

cluster nodes.

Hive用户自定义反射

你可以使用Java中的类或者方法来执行一个功能，那么这些类和方法将使用Java的反射执行，你可以调用这些类的static方法。方法将返回一个私有类型，或者一个hive知道怎样去Serialize(序列化)的类型。

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1704359)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. SELECT reflect("java.lang.String", "valueOf", 1),
2. reflect("java.lang.String", "isEmpty"),
3. reflect("java.lang.Math", "max", 2, 3),
4. reflect("java.lang.Math", "min", 2, 3),
5. reflect("java.lang.Math", "round", 2.5),
6. reflect("java.lang.Math", "exp", 1.0),
7. reflect("java.lang.Math", "floor", 1.9)
8. FROM src LIMIT 1;
9. 1 true 3 2 3 2.7182818284590455 1.0
10. 数据操作语句

一、创建表的语法

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1703702)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table\_name
2. [(col\_name data\_type [COMMENT col\_comment], ...)]
3. [COMMENT table\_comment]
4. [PARTITIONED BY (col\_name data\_type [COMMENT col\_comment], ...)]
5. [CLUSTERED BY (col\_name, col\_name, ...) [SORTED BY (col\_name [ASC|DESC], ...)] INTO num\_buckets BUCKETS]
6. [ROW FORMAT row\_format]
7. [STORED AS file\_format]
8. [LOCATION hdfs\_path]
9. [TBLPROPERTIES (property\_name=property\_value, ...)]
10. [AS select\_statement]
11. CREATE [EXTERNAL] TABLE [IF NOT EXISTS] table\_name
12. LIKE existing\_table\_name
13. [LOCATION hdfs\_path]
14. data\_type
15. : primitive\_type
16. | array\_type
17. | map\_type
18. | struct\_type
19. primitive\_type
20. : TINYINT
21. | SMALLINT
22. | INT
23. | BIGINT
24. | BOOLEAN
25. | FLOAT
26. | DOUBLE
27. | STRING
28. array\_type
29. : ARRAY < data\_type >
30. map\_type
31. : MAP < primitive\_type, data\_type >
32. struct\_type
33. : STRUCT < col\_name : data\_type [COMMENT col\_comment], ...>
34. row\_format
35. : DELIMITED [FIELDS TERMINATED BY char] [COLLECTION ITEMS TERMINATED BY char]
36. [MAP KEYS TERMINATED BY char] [LINES TERMINATED BY char]
37. | SERDE serde\_name [WITH SERDEPROPERTIES (property\_name=property\_value, property\_name=property\_value, ...)]
38. file\_format:
39. : SEQUENCEFILE
40. | TEXTFILE
41. | INPUTFORMAT input\_format\_classname OUTPUTFORMAT output\_format\_classname

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

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

[COMMENT table\_comment]

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

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

[ROW FORMAT row\_format]

[STORED AS file\_format]

[LOCATION hdfs\_path]

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

[AS select\_statement]

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

LIKE existing\_table\_name

[LOCATION hdfs\_path]

data\_type

: primitive\_type

| array\_type

| map\_type

| struct\_type

primitive\_type

: TINYINT

| SMALLINT

| INT

| BIGINT

| BOOLEAN

| FLOAT

| DOUBLE

| STRING

array\_type

: ARRAY < data\_type >

map\_type

: MAP < primitive\_type, data\_type >

struct\_type

: STRUCT < col\_name : data\_type [COMMENT col\_comment], ...>

row\_format

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

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

| SERDE serde\_name [WITH SERDEPROPERTIES (property\_name=property\_value, property\_name=property\_value, ...)]

file\_format:

: SEQUENCEFILE

| TEXTFILE

| INPUTFORMAT input\_format\_classname OUTPUTFORMAT output\_format\_classname

用已知表名创建一个表的时候，如果同名表已经存在则会报一个表已存在的异常，不过你可以使用IF NOT EXISTS来防止这个错误。  
  
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  
  
用户在建表的时候可以自定义 SerDe 或者使用自带的 SerDe。如果没有指定 ROW FORMAT 或者 ROW FORMAT DELIMITED，将会使用自带的 SerDe。在建表的时候，用户还需要为表指定列，用户在指定表的列的同时也会指定自定义的 SerDe，Hive 通过 SerDe 确定表的具体的列的数据。  
  
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 SEQUENCE 。使用INPUTFROMAT 和OUTPUTFORMAT将指定输入输出的格式，例如：  
  
'org.apache.hadoop.hive.contrib.fileformat.base64.Base64TextInputFormat'.  
  
代表指定了表的输入格式必须的是Base64.  
  
有分区的表可以在创建的时候使用 PARTITIONED BY 语句。一个表可以拥有一个或者多个分区，每一个分区单独存在一个目录下。而且，表和分区都可以对某个列进行 CLUSTERED BY 操作，将若干个列放入一个桶（bucket）中。也可以利用SORT BY 对数据进行排序。这样可以为特定应用提高性能。  
  
表名和列名不区分大小写，SerDe 和属性名区分大小写。表和列的注释是字符串。  
  
下面是创建一个表的例子:

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1703702)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. CREATE TABLE page\_view(viewTime INT, userid BIGINT,
2. page\_url STRING, referrer\_url STRING,
3. ip STRING COMMENT 'IP Address of the User')
4. COMMENT 'This is the page view table'
5. PARTITIONED BY(dt STRING, country STRING)
6. STORED AS SEQUENCEFILE;

CREATE TABLE page\_view(viewTime INT, userid BIGINT,

page\_url STRING, referrer\_url STRING,

ip STRING COMMENT 'IP Address of the User')

COMMENT 'This is the page view table'

PARTITIONED BY(dt STRING, country STRING)

STORED AS SEQUENCEFILE;

创建一个page\_view表，按照dt和country进行分割并有序排列。  
  
二、使用select来创建表（简称CTAS）  
表可以按照查询结果来进行创建，按照CTAS查询的来的表是自动的，这意味着这个表不会被其他人看见值到查询结果结束，其他人只能看见查询的来的完成结果，而不是看到这个表的全部。  
  
CTAS语句分为2个部分，HiveQL支持SELECT部分，创建部分将会使用SELECT查询的结果，并且可以使用其他表的属性例如SerDe和存储格式来创建这个表。CTAS创建表的唯一限制是不能够创建一个分区表，也不能创建一个EXTERNAL的表。  
  
三、使用 SerDes  
SerDe 是 Serialize/Deserilize 的简称，目的是用于序列化和反序列化。序列化的格式包括：  
• 分隔符（tab、逗号、CTRL-A）   
• Thrift 协议   
反序列化（内存内）：  
• Java Integer/String/ArrayList/HashMap   
• Hadoop Writable 类   
• 用户自定义类   
  
  
CTAS创建如果没有指定列名那么查询得到的列名将会被自动分配为 \_col0, \_col1, and \_col2等这样的列名。另外，新的目标表将会使用特殊的SerDe进行创建，并且存储格式与查询语句相独立。

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1703702)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. CREATE TABLE new\_key\_value\_store
2. ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"
3. STORED AS RCFile AS
4. SELECT (key % 1024) new\_key, concat(key, value) key\_value\_pair
5. FROM key\_value\_store
6. SORT BY new\_key, key\_value\_pair;

CREATE TABLE new\_key\_value\_store

ROW FORMAT SERDE "org.apache.hadoop.hive.serde2.columnar.ColumnarSerDe"

STORED AS RCFile AS

SELECT (key % 1024) new\_key, concat(key, value) key\_value\_pair

FROM key\_value\_store

SORT BY new\_key, key\_value\_pair;

把一个表中的数据查询出来得到另外一个表示Hive一个重要的Feature，在这个查询过程中，你可以让Hive处理将原数据格式转化为另外的一种格式。  
  
四、按BUCKET（桶）存储的表

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1703702)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. CREATE TABLE page\_view(viewTime INT, userid BIGINT,
2. page\_url STRING, referrer\_url STRING,
3. ip STRING COMMENT 'IP Address of the User')
4. COMMENT 'This is the page view table'
5. PARTITIONED BY(dt STRING, country STRING)
6. CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS
7. ROW FORMAT DELIMITED
8. FIELDS TERMINATED BY '\001'
9. COLLECTION ITEMS TERMINATED BY '\002'
10. MAP KEYS TERMINATED BY '\003'
11. STORED AS SEQUENCEFILE;

CREATE TABLE page\_view(viewTime INT, userid BIGINT,

page\_url STRING, referrer\_url STRING,

ip STRING COMMENT 'IP Address of the User')

COMMENT 'This is the page view table'

PARTITIONED BY(dt STRING, country STRING)

CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\001'

COLLECTION ITEMS TERMINATED BY '\002'

MAP KEYS TERMINATED BY '\003'

STORED AS SEQUENCEFILE;

上面的例子中，page\_view 表按照userid进行分桶，共32个桶，数据按照viewTime进行排序。  
意义在于，这样可以对数据集群按照userid做一个有效的抽样结果。排序属性可以让开发人员更好的利用已知的数据结构进行查询，当某一个列是lists或者maps类型时，还可以使用MAP KEYS 和 COLLECTION TIEMS关键字。  
  
CLUSTERED BY 和 SORTERD BY不影响数据插入方式，只影响读取顺序。这意味着用户必须小心使用这2个命令，看看桶类型表是怎么工作的吧。  
  
五、外部表  
External Tables  
  
External Table 指向已经在 HDFS 中存在的数据，可以创建 Partition。它和 Table 在元数据的组织上是相同的，而实际数据的存储则有较大的差异

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1703702)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. CREATE EXTERNAL TABLE page\_view(viewTime INT, userid BIGINT,
2. page\_url STRING, referrer\_url STRING,
3. ip STRING COMMENT 'IP Address of the User',
4. country STRING COMMENT 'country of origination')
5. COMMENT 'This is the staging page view table'
6. ROW FORMAT DELIMITED FIELDS TERMINATED BY '\054'
7. STORED AS TEXTFILE
8. LOCATION '<hdfs\_location>';

CREATE EXTERNAL TABLE page\_view(viewTime INT, userid BIGINT,

page\_url STRING, referrer\_url STRING,

ip STRING COMMENT 'IP Address of the User',

country STRING COMMENT 'country of origination')

COMMENT 'This is the staging page view table'

ROW FORMAT DELIMITED FIELDS TERMINATED BY '\054'

STORED AS TEXTFILE

LOCATION '<hdfs\_location>';

六、使用Like来创建表  
like仅复制表结构，但是不复制数据

Java代码 [复制代码](http://myeyeofjava.iteye.com/blog/1703702)[[收藏代码http://myeyeofjava.iteye.com/images/spinner.gif](javascript:void())](javascript:void())

1. CREATE TABLE empty\_key\_value\_store
2. LIKE key\_value\_store;

CREATE TABLE empty\_key\_value\_store

LIKE key\_value\_store;

七、drop  
删除一个内部表的同时会同时删除表的元数据和数据。删除一个外部表，只删除元数据而保留数据。

# Hive 优化

**一、 控制hive任务中的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(目前为128M, 可在hive中通过set dfs.block.size;命令查看到，该参数不能自定义修改)；  
  
2. 举例：  
a) 假设input目录下有1个文件a,大小为780M,那么hadoop会将该文件a分隔成7个块（6个128m的块和1个12m的块），从而产生7个map数  
b) 假设input目录下有3个文件a,b,c,大小分别为10m，20m，130m，那么hadoop会分隔成4个块（10m,20m,128m,2m）,从而产生4个map数  
即，如果文件大于块大小(128m),那么会拆分，如果小于块大小，则把该文件当成一个块。  
  
3. 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。

4. 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3和4，我们需要采取两种方式来解决：即减少map数和增加map数；  
  
**如何合并小文件，减少map数？**  
假设一个SQL任务：  
Select count(1) from popt\_tbaccountcopy\_mes where pt = ‘2012-07-04’;  
该任务的inputdir /group/p\_sdo\_data/p\_sdo\_data\_etl/pt/popt\_tbaccountcopy\_mes/pt=2012-07-04  
共有194个文件，其中很多是远远小于128m的小文件，总大小9G，正常执行会用194个map任务。  
Map总共消耗的计算资源： SLOTS\_MILLIS\_MAPS= 623,020  
  
我通过以下方法来在map执行前合并小文件，减少map数：  
set mapred.max.split.size=100000000;  
set mapred.min.split.size.per.node=100000000;  
set mapred.min.split.size.per.rack=100000000;  
set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;  
再执行上面的语句，用了74个map任务，map消耗的计算资源：SLOTS\_MILLIS\_MAPS= 333,500  
对于这个简单SQL任务，执行时间上可能差不多，但节省了一半的计算资源。  
大概解释一下，100000000表示100M, set hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat;这个参数表示执行前进行小文件合并，  
前面三个参数确定合并文件块的大小，大于文件块大小128m的，按照128m来分隔，小于128m,大于100m的，按照100m来分隔，把那些小于100m的（包括小文件和分隔大文件剩下的），  
进行合并,最终生成了74个块。  
  
**如何适当的增加map数？**  
  
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。  
假设有这样一个任务：  
Select data\_desc,  
count(1),  
count(distinct id),  
sum(case when …),  
sum(case when ...),  
sum(…)  
from a group by data\_desc  
如果表a只有一个文件，大小为120M，但包含几千万的记录，如果用1个map去完成这个任务，肯定是比较耗时的，这种情况下，我们要考虑将这一个文件合理的拆分成多个，  
这样就可以用多个map任务去完成。  
set mapred.reduce.tasks=10;  
create table a\_1 as   
select \* from a   
distribute by rand(123);   
  
这样会将a表的记录，随机的分散到包含10个文件的a\_1表中，再用a\_1代替上面sql中的a表，则会用10个map任务去完成。  
每个map任务处理大于12M（几百万记录）的数据，效率肯定会好很多。  
  
看上去，貌似这两种有些矛盾，一个是要合并小文件，一个是要把大文件拆成小文件，这点正是重点需要关注的地方，  
根据实际情况，控制map数量需要遵循两个原则：使大数据量利用合适的map数；使单个map任务处理合适的数据量；

**二、 控制hive任务的reduce数：**  
  
1. Hive自己如何确定reduce数：  
reduce个数的设定极大影响任务执行效率，不指定reduce个数的情况下，Hive会猜测确定一个reduce个数，基于以下两个设定：  
hive.exec.reducers.bytes.per.reducer（每个reduce任务处理的数据量，默认为1000^3=1G）   
hive.exec.reducers.max（每个任务最大的reduce数，默认为999）  
计算reducer数的公式很简单N=min(参数2，总输入数据量/参数1)  
即，如果reduce的输入（map的输出）总大小不超过1G,那么只会有一个reduce任务；  
如：select pt,count(1) from popt\_tbaccountcopy\_mes where pt = '2012-07-04' group by pt;   
/group/p\_sdo\_data/p\_sdo\_data\_etl/pt/popt\_tbaccountcopy\_mes/pt=2012-07-04 总大小为9G多，因此这句有10个reduce  
  
2. 调整reduce个数方法一：  
调整hive.exec.reducers.bytes.per.reducer参数的值；  
set hive.exec.reducers.bytes.per.reducer=500000000; （500M）  
select pt,count(1) from popt\_tbaccountcopy\_mes where pt = '2012-07-04' group by pt; 这次有20个reduce  
  
3. 调整reduce个数方法二；  
set mapred.reduce.tasks = 15;  
select pt,count(1) from popt\_tbaccountcopy\_mes where pt = '2012-07-04' group by pt;这次有15个reduce  
  
4. reduce个数并不是越多越好；  
同map一样，启动和初始化reduce也会消耗时间和资源；  
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；  
  
5. 什么情况下只有一个reduce；  
很多时候你会发现任务中不管数据量多大，不管你有没有设置调整reduce个数的参数，任务中一直都只有一个reduce任务；  
其实只有一个reduce任务的情况，除了数据量小于hive.exec.reducers.bytes.per.reducer参数值的情况外，还有以下原因：  
a) 没有group by的汇总，比如把select pt,count(1) from popt\_tbaccountcopy\_mes where pt = '2012-07-04' group by pt; 写成 select count(1) from popt\_tbaccountcopy\_mes where pt = '2012-07-04';  
这点非常常见，希望大家尽量改写。  
b) 用了Order by  
c) 有笛卡尔积  
通常这些情况下，除了找办法来变通和避免，我暂时没有什么好的办法，因为这些操作都是全局的，所以hadoop不得不用一个reduce去完成；  
  
同样的，在设置reduce个数的时候也需要考虑这两个原则：使大数据量利用合适的reduce数；使单个reduce任务处理合适的数据量；

Hive是将符合SQL语法的字符串解析生成可以在Hadoop上执行的MapReduce的工具。

使用Hive尽量按照分布式计算的一些特点来设计sql，和传统关系型数据库有区别，

所以需要去掉原有关系型数据库下开发的一些固有思维。

基本原则：

1：尽量尽早地过滤数据，减少每个阶段的数据量,对于分区表要加分区，同时只选择需要使用到的字段

select ... from A

join B

on A.key = B.key

where A.userid>10

and B.userid<10

and A.dt='20120417'

and B.dt='20120417';

应该改写为：

select .... from (select .... from A

where dt='201200417'

and userid>10

) a

join ( select .... from B

where dt='201200417'

and userid < 10

) b

on a.key = b.key;

2：尽量原子化操作，尽量避免一个SQL包含复杂逻辑

可以使用中间表来完成复杂的逻辑

drop table if exists tmp\_table\_1;

create table if not exists tmp\_table\_1 as

select ......;

drop table if exists tmp\_table\_2;

create table if not exists tmp\_table\_2 as

select ......;

drop table if exists result\_table;

create table if not exists result\_table as

select ......;

drop table if exists tmp\_table\_1;

drop table if exists tmp\_table\_2;

3：单个SQL所起的JOB个数尽量控制在5个以下

4：慎重使用mapjoin,一般行数小于2000行，大小小于1M(扩容后可以适当放大)的表才能使用,小表要注意放在join的左边（目前TCL里面很多都小表放在join的右边）。

否则会引起磁盘和内存的大量消耗

5：写SQL要先了解数据本身的特点，如果有join ,group操作的话，要注意是否会有数据倾斜

如果出现数据倾斜，应当做如下处理：

set hive.exec.reducers.max=200;

set mapred.reduce.tasks= 200;---增大Reduce个数

set hive.groupby.mapaggr.checkinterval=100000 ;--这个是group的键对应的记录条数超过这个值则会进行分拆,值根据具体数据量设置

set hive.groupby.skewindata=true; --如果是group by过程出现倾斜 应该设置为true

set hive.skewjoin.key=100000; --这个是join的键对应的记录条数超过这个值则会进行分拆,值根据具体数据量设置

set hive.optimize.skewjoin=true;--如果是join 过程出现倾斜 应该设置为true

6：如果union all的部分个数大于2，或者每个union部分数据量大，应该拆成多个insert into 语句，实际测试过程中，执行时间能提升50%

insert overwite table tablename partition (dt= ....)

select ..... from (

select ... from A

union all

select ... from B

union all

select ... from C

) R

where ...;

可以改写为：

insert into table tablename partition (dt= ....)

select .... from A

WHERE ...;

insert into table tablename partition (dt= ....)

select .... from B

WHERE ...;

insert into table tablename partition (dt= ....)

select .... from C

WHERE ...;