# Hive

## hive简介

Hive是一个基于Hadoop的开源数据仓库工具，用于存储和处理海量结构化数据。它把海量数据存储于Hadoop文件系统，而不是数据库，但提供了一套类数据库的数据存储和处理机制，并采用类SQL语言对这些数据进行自动化管理和处理。我们可以把Hive中海量结构化数据看成一个个的表，而实际上这些数据是分布式存储在HDFS 中的。Hive经过对语句进行解析和转换，最终生成一系列基于Hadoop 的MapReduce任务，通过执行这些任务完成数据处理。

使用Hive的命令行接口，感觉很像操作关系数据库，但是Hive和关系数据库还是有很大的不同，具体总结如下：

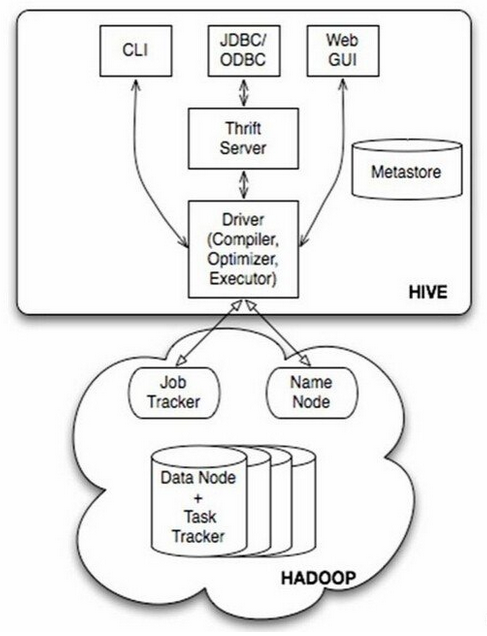
1.是存储文件的系统不同。Hive使用的是Hadoop的HDFS，关系数据库则是服务器本地的文件系统。

2.计算模型不同。Hive使用MapReduce计算模型，而关系数据库则是自己设计的计算模型。

3.设计目的不同。关系数据库都是为实时查询的业务设计的，而Hive则是为海量数据做数据挖掘设计的，实时性很差。实时性的区别导致Hive的应用场景和关系数据库有很大的不同。

4.扩展能力不同。Hive通过集成Hadoop使得很容易扩展自己的存储能力和计算能力，而关系数据库在这个方面要比数据库差很多。

Hive的技术架构如下图所示。



由上图可知，Hadoop和MapReduce是Hive架构的根基。Hive架构包括如下组件：CLI（command line interface）、JDBC/ODBC、Thrift Server、Web GUI、Metastore和Driver（Complier、Optimizer和Executor），这些组件可分为服务端组件和客户端组件两大类。

**服务端组件：**

**Driver组件：**包括Complier、Optimizer和Executor，它的作用是将我们写的HiveQL语句进行解析、编译优化，生成执行计划，然后调用底层的MapReduce计算框架。

**Metastore组件：**是元数据服务组件，它存储Hive的元数据，Hive的元数据存储在关系数据库里，支持的关系数据库有derby和mysql。元数据对于Hive十分重要，因此**Hive支持把Metastore服务独立出来，安装到远程的服务器集群里**，从而解耦Hive服务和Metastore服务，保证Hive运行的健壮性。

**Thrift Server：**是Facebook开发的一个软件框架，它用来进行可扩展且跨语言的服务的开发，Hive集成了该服务，能让不同的编程语言调用Hive的接口。

**客户端组件：**

1.CLI即command lineinterface，命令行接口。

2.JDBC/ODBC 是Thrift的客户端。

3.Web GUI是Hive客户端提供的一种通过网页的方式访问Hive的服务。这个接口对应Hive的hwi组件，使用前要启动hwi服务。

**Hive内置服务**

Hive自带了许多服务，可在运行时通过service选项来明确指定使用什么服务，或通过--service help来查看帮助。下面介绍最常用的一些服务。

（1）CLI：这是Hive的命令行界面，用的比较多。这是默认的服务，直接可以在命令行里面使用。

（2）hiveserver：这个可以让Hive以提供Trift服务的服务器形式来运行，可以允许许多不同语言编写的客户端进行通信。可以通过设置HIVE\_PORT环境变量来设置服务器所监听的端口号，在默认的情况下，端口为 10000。最新版本（hive1.2.1）用**hiveserver2**取代了原有的hiveserver。

（3）hwi：它是Hive的Web接口，是hive cli的一个web替换方案。

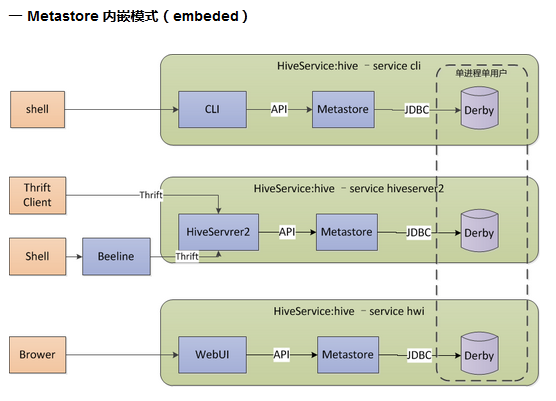
（4）jar：与Hadoop jar等价的Hive接口，这是运行类路径中同时包含Hadoop和Hive类的Java应用程序的简便方式。

（5）Metastore：用于连接元数据库（如mysql）。在默认情况下，Metastore和Hive服务运行在同一个进程中，端口号为9083。使用这个服务，可以让Metastore作为一个单独的进程运行，我们可以通过METASTORE\_PORT来指定监听的端口号。

## Metastore部署模式

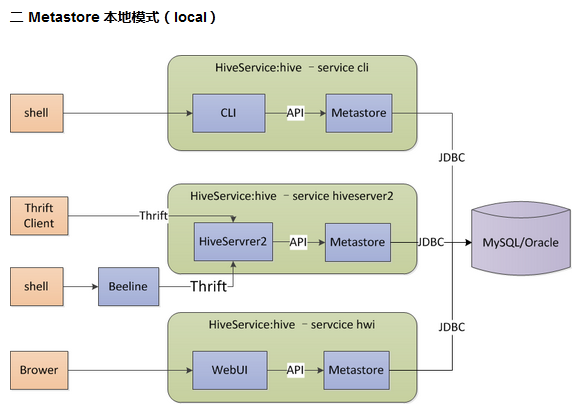
**1 内嵌模式**

内嵌模式使用内嵌的Derby数据库存储元数据，只能单用户操作，一般用于单元测试。其架构图如下所示。



**2 本地模式**

本地模式与内嵌模式最大的区别在于**数据库由内嵌于hive服务变成独立部署（一般为mysql数据库）**，**hive服务使用jdbc访问元数据**，多个服务可以同时访问。mysql数据库用于存储元数据，可安装在本地或远程服务器上，在配置文件hive-site.xml中指定jdbc URL、驱动、用户名、密码等属性。其中属性hive.metastore.uris的值为空，表示为嵌入模式或本地模式。在本地模式中，每种hive服务（如cli、hiveserver2、hwi）都内置启动了一个metastore服务，用于连接mysql元数据库。其架构图如下所示。



**3 远程模式**

远程模式将原内嵌于hive服务中的**metastore服务独立出来单独运行，hive服务通过thrift访问metastore**，这种模式可以控制到数据库的连接等。其中，metastore服务器需要通过hive-site.xml配置jdbc URL、驱动、用户名、密码等属性，hiveserver2服务器和cli客户端需要通过hive-site.xml配置hive.metastore.uris属性，用于指定metastore服务地址（如thrift://localhost:9083），metastore服务器通过./hive --service metastore开启metastore服务，hiveserver2服务器通过./hive --service hiveserver2开启hiveserver服务，客户端通过./hive shell 或./beeline进行连接。其架构图如下所示。

远程模式下可以按如下部署规划：

（1）元数据服务器：部署metastore服务和mysql数据库。

（2）hiveserver服务器：用于部署hiveserver2服务，可通过thrift访问metastore。

（3）客户服务器：部署hive客户端，可以基于cli、beeline或直接使用thrift访问hiveserver2。

hive中的表

1.managed table 托管表：删除表的时候把数据也删除

2.外部表：删除表的时候数据不会删除

## 创建表

create table student\_info(

student\_id string comment '学号',

name string comment '姓名',

age int comment '年龄',

origin string comment '地域'

)

comment '学生信息表'

row format delimited

fields terminated by '\t' **-*-指定分割符***

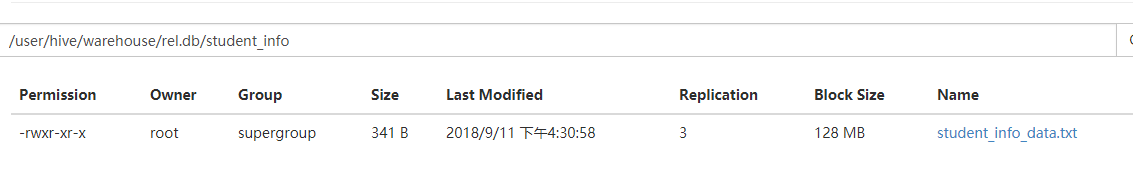
lines terminated by '\n' ***--指定换行符***

stored as textfile;

## 导入数据

### 从本地导入

**load data local inpath** ‘<local path>' **into table** <table\_name>;



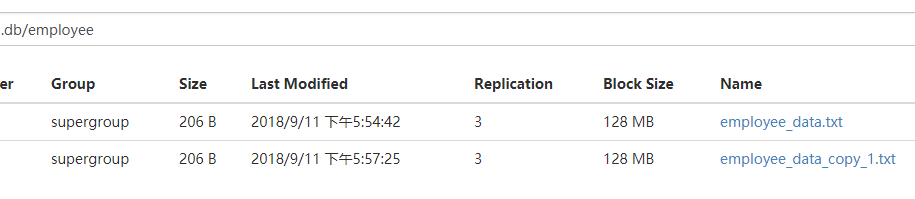
导入之后，在数据库的该表(student\_info)路径下会增加上传的文件，这是内部管理表的情况。

查询语句跟sql类似。

从本地导入多次数据到同一个表中，即使是相同的文件也不会覆盖，而是产生一个副本文件，然后追加到表末尾。

如果需要**覆盖原表的数据**，使用**overwrite关键字**。

**load data local inpath** ‘<local path>' **overwrite** **into table** <table\_name>;



employee表中导入两个相同的文件。

### 从HDFS中导入

**load data inpath** ‘<hdfs path>' **into table** <table\_name>;

**load data inpath** ‘<hdfs path>' **overwirte** **into table** <table\_name>;

overwrite会把hdfs的文件**剪切**到表目录下。（区分本地模式）

## 创建外部表

外部表：删除表的时候不会把数据删除。

可以提前创建好hdfs路径

hadoop mkdir -p /user/hive/warehouse/data/student\_school\_info

如果没有提前创建好，在创建外部表的时候会根据指定路径自动创建。

创建外部表学生入学信息

字段信息：

学号、姓名、学院id、专业id、入学年份

HDFS数据路径：/user/hive/warehouse/data/student\_school\_info

create **external** table rel.student\_school\_info(

student\_id string,

name string,

institute\_id string,

major\_id string,

school\_year string

)

row format delimited

fields terminated by '\t'

lines terminated by '\n'

stored as textfile

**location '/user/hive/warehouse/data/student\_school\_info'; --指定hdfs中的外部存储目录**

## 分区表

**从目录的层面控制搜索数据的范围。**

### 内部分区表

#### 创建表

create table student\_school\_info\_partition(

student\_id string,

name string,

institute\_id string,

major\_id string

)

**partitioned by(school\_year string) --在建表时就指定分区**

row format delimited

fields terminated by '\t'

lines terminated by '\n'

stored as textfile;

**查看分区**

show **partitions** student\_school\_info\_partition;

#### 从已有的表中导入数据

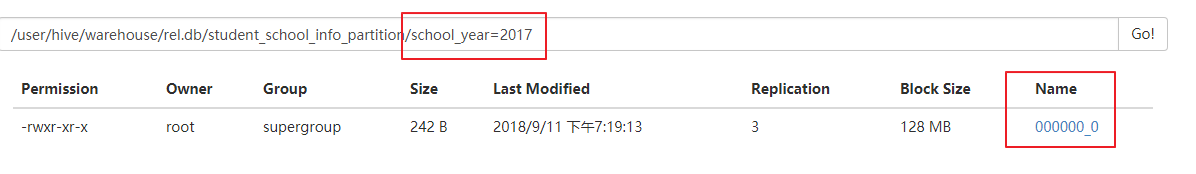
使用insert into从student\_school\_info表（此表中有school\_year字段）将2017年入学的学籍信息导入到student\_school\_info\_partition分区表中（自动创建目录）

insert into table student\_school\_info\_partition partition(school\_year='2017')

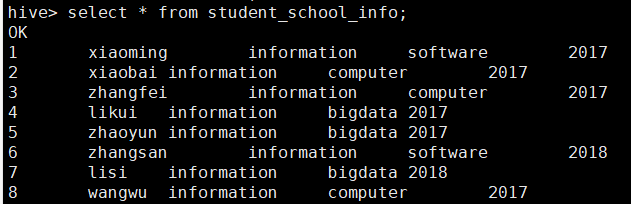
select t1.student\_id,t1.name,t1.institute\_id,t1.major\_id

from student\_school\_info t1

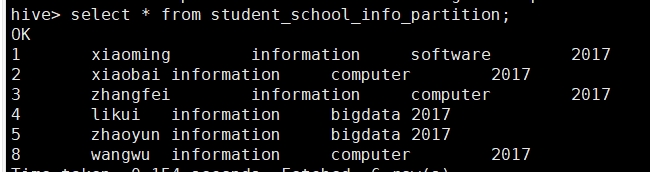
where t1.school\_year=2017;

查看hdfs路径，增加分区目录和分区后的文件，文件中存放的是分区后内容。

原始表(student\_school\_info)

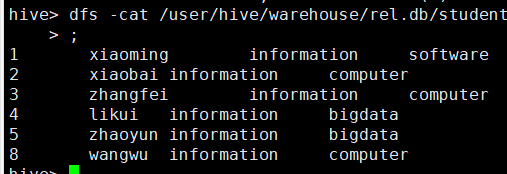


分区表(student\_school\_info\_partition)

  
select \* from student\_school\_info\_partition where school\_year='2017';

查询分区表时，把字段映射成目录 / student\_school\_info\_partition/2017

经过mr后的输出文件000000\_0



**删除分区（分区删除之后，数据也会被删除，但是，表结构中依然存在这个分区(分区实际上也是表的字段)，只是文件目录下的文件夹被删除了，使用show partitions时也查找不出来）**

alter table student\_school\_info\_partition **drop partition (school\_year='2017');**

**添加分区,创建目录，使用时，应该注意格式要按照创建表结构时的创建的分区字段，并不是真正的创建新的分区字段，而只是在文件路径中创建新的目录**

alter table student\_school\_info\_partition **add partition (school\_year='2018');**

//在文件路径中创建 ……/year=2018的目录

#### 使用动态分区添加数据

set hive.exec.dynamic.partition=true;

set hive.exec.dynamic.partition.mode=**nonstrict**; --非严格模式

insert overwrite table student\_school\_info\_partition partition(school\_year)

select t1.student\_id,t1.name,t1.institute\_id,t1.major\_id,t1.school\_year

from student\_school\_info t1

**动态分区**：不在执行语句中指定分区，**通过查询结果自动创建分区**开启动态分区功能（在路径中自动创建目录）。如果是严格模式，至少要指定一个静态分区，非严格模式下不需要指定静态分区。

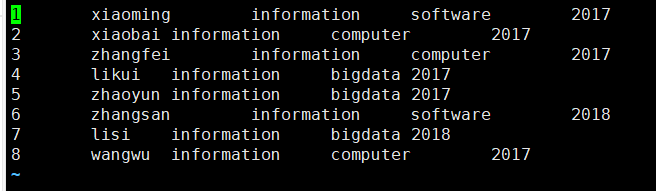
**静态分区**，就是在插入数据的时候指定了分区字段的值，所有的数据将会插入到该分区中。

#### 从本地中导入数据

首先分区的文件夹应该存在，然后导入到相应的分区目录

load data local inpath ‘local path’ into table table\_name where school\_year=’2017’;

导入的数据格式应该和表结构一样，如果分区字段的对应数据中不等于设定的分区值，会自动变换成分区值。



例如，上表中有两个分区（school\_year=2017和2018），但是原始数据的school\_year字段的值都是2017，如果把原始文件导入到2018分区，那么school\_year字段的值将会**全部设置**为2018.

### 创建外部分区表

1.创建学生入学信息表

字段信息：学号、姓名、学院id、专业id

分区字段：入学年份

create external table rel.student\_school\_info\_external\_partition(

student\_id string,

name string,

institute\_id string,

major\_id string

)

**partitioned by(school\_year string)**

row format delimited

fields terminated by '\t'

lines terminated by '\n'

stored as textfile

**location '/user/hive/warehouse/data/student\_school\_info\_external\_partition'; --指定路径**

2 在分区表的hdfs路径中添加school\_year=2017目录

3.手动添加分区

alter table student\_school\_info\_external\_partition **add partition(school\_year='2017');**

查看和删除分区的方法和内部分区相同。

删除分区之后，数据依然存在。

## 查看数据

hive中的普通查询不会进行mr，而连接查询会进行mr操作。使用聚合函数会进行mr操作。

### 排序

* **order：全排序。**

在reduce端会进行排序，hive会自动把reduce任务数设置为1，即使用户显式设置了其他值。A sorted order is maintained across all of the output from every reducer。这个排序是对于所有reduce的输出有序的。

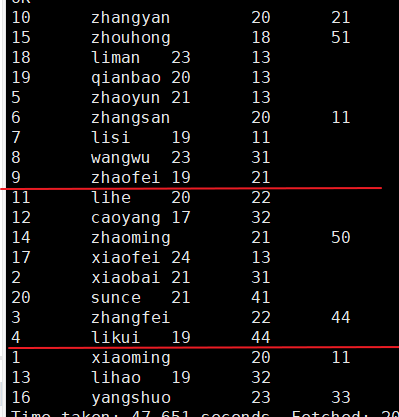
select \* from orders order by id asc ;

* **sort by排序。**

map端排序,本地有序，在每个reduce任务中是有序的，但是整体是无序的。

(This indicates which columns to sort when ordering the reducer input records. This means it completes sorting before sending data to the reducer. )

$hive>select \* from orders sort by id asc ;



设置了3个reduce任务，经过sort后，每个reduce中都是有序的，但整体无序。

* **distribute by**

**类似于mysql的group by,进行分区操作**。

根据distribute设置的字段决定map的输出进入哪个reduce中。实际上就是一个partition的作用。需要排在sort by前面 distribute by ...... sort by ..

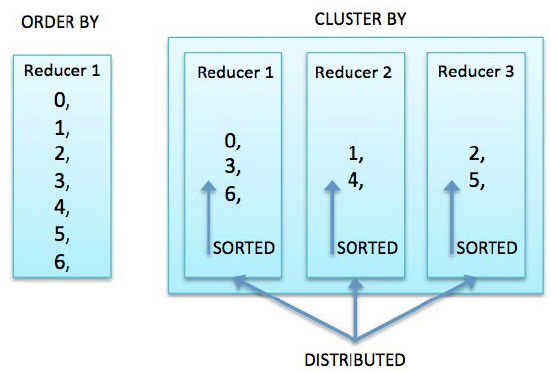
select id,orderno,cid from orders distribute by cid sort by cid desc ;

* **cluster by**

它是 distribute by .. sort by .. 的简写方式。

**the CLUSTER BY operation is sorted in each distributed group**

如果想要进行全排序，那么要先 cluster by 再 order by



order by 和 cluster by 的区别

## 分桶表

按照指定字段取它的hash散列值分桶

创建学生入学信息分桶表

字段信息：学号、姓名、学院ID、专业ID

分桶字段：学号，4个桶，桶内按照学号升序排列

create table rel.student\_info\_bucket(

student\_id string,

name string,

age int,

origin string

)

**clustered by (student\_id) sorted by (student\_id asc) into 4 buckets**

row format delimited

fields terminated by '\t'

lines terminated by '\n'

stored as textfile;

**向student\_info\_bucket分桶表插入数据**

set hive.enforce.bucketing = true;

set mapreduce.job.reduces=4;

insert overwrite table student\_info\_bucket

select student\_id,name,age,origin

from student\_info

cluster by(student\_id);

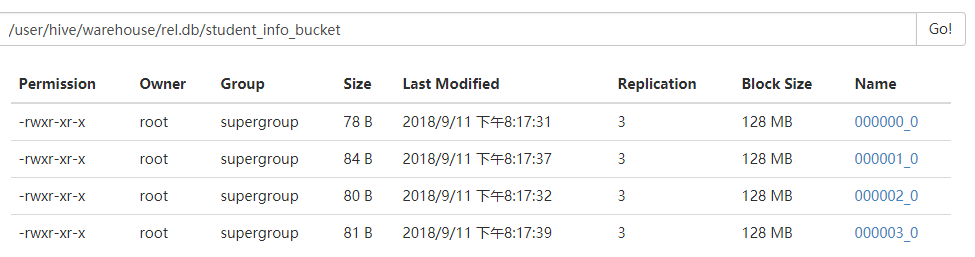
**此时，student\_info\_bucket 中的数据是按照分桶排列的。**

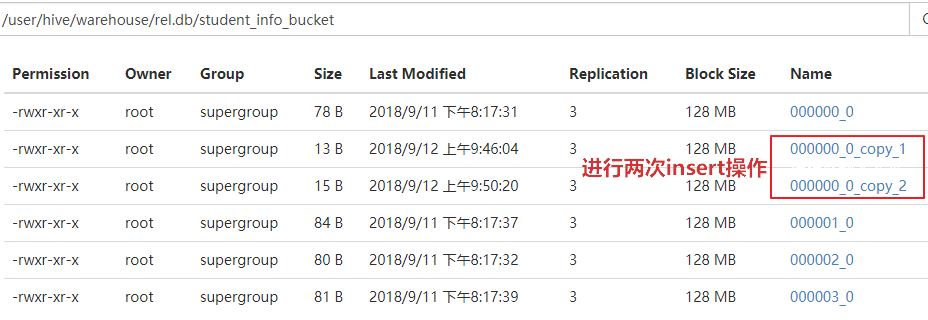
**hdfs的路径中每个桶存放在一个文件中。**

分桶表一般不使用load向分桶表中导入数据，**因为load导入数据只是将数据复制到表的数据存储目录下，hive并不会在load的时候对数据进行分析然后按照分桶字段分桶**，load只会将一个文件全部导入到分桶表中，并没有分桶。一般采用insert从其他表向分桶表插入数据。

分桶表在创建表的时候只是定义表的模型，插入的时候需要做如下操作：

在每次执行分桶插入的时候在当前执行的session会话中要设置hive.enforce.bucketing = true;声明本次执行的是一次分桶操作。需要指定reduce个数与分桶的数量相同set mapreduce.job.reduces=4，这样才能保证有多少桶就生成多少个文件。如果定义了按照分桶字段排序，需要在从其他表查询数据过程中将数据按照分区字段排序之后插入各个桶中，分桶表并不会将各分桶中的数据排序。排序和分桶的字段相同的时候使用Cluster by(字段),cluster by 默认按照分桶字段在桶内升序排列，如果需要在桶内降序排列，使用distribute by (col) sort by (col desc)组合实现



对表进行单条数据的insert操作，会重新进行mr操作，并生成新的输出文件，但是桶的个数是不变的。如下图，新增的两个copy文件仍然属于第一个桶，只是进行了分段。

**桶表的数量如何设置?**

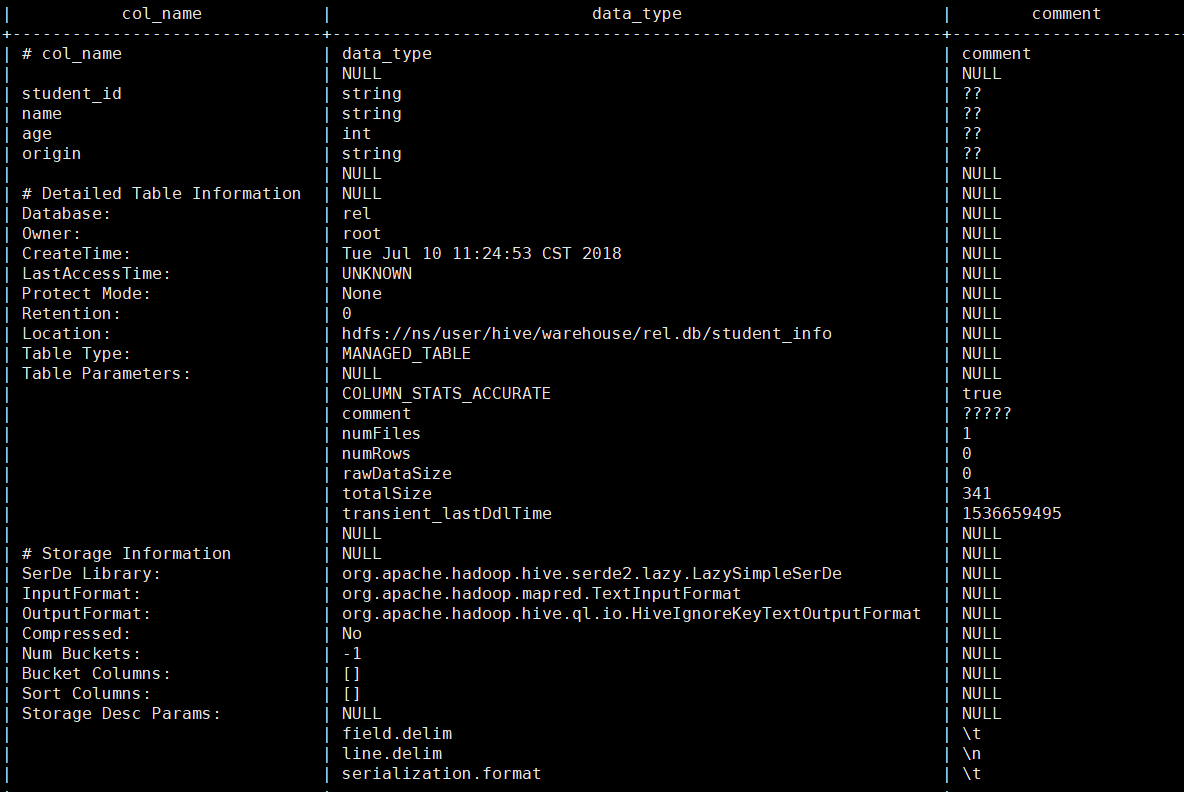
//评估数据量，保证每个桶的数据量block的2倍大小。

## 表级操作

查看表结构

**desc** table; //查看简单的表信息，如字段

**desc** formatted table; //查看详细的表信息



**1.根据已存在的表结构，使用like关键字，复制一个表结构一模一样的新表。不带数据，只创建表结构。**

create table **copy\_table** like **origin\_table**;

**2.根据已经存在的表，使用as关键字，创建一个与查询结果字段一致的表，同时将查询结果数据插入到新表。携带数据和表结构**

create table **new\_table** as select \* from origin\_table;

只有student\_id,name两个字段的表

create table **new\_table** as select student\_id,name from **origin\_table**;

**3.old\_table表重命名为new\_table**

alter table **old\_table** rename to **old\_table**;

**4.给orgin\_table表添加new\_field列,新添加的字段会在所有列最后，分区列之前，在添加新列之前已经存在的数据文件中**

如果没有新添加列对应的数据，在查询的时候显示为空。添加多个列用逗号隔开。

alter table **orgin\_table** add columns (**new\_field** string comment '性别');

**5. 修改列，将继续存在的列再定义一遍，需要替换的列重新定义**

alter table **orgin\_table** replace columns(student\_id string,name string,age int,origin string,gender2 int);

**6.删除列,将继续存在的列再定义一遍，需要删除的列不再定义**

alter table student\_info\_new3 replace columns(student\_id string,name string,age int,origin string);

**7.启用/禁用表**

$hive>ALTER TABLE t2 **ENABLE NO\_DROP**; //不允许删除

$hive>ALTER TABLE t2 **DISABLE NO\_DROP**; //允许删除

## 导出数据

使用insert将student\_info**表数据**导出到本地指定路径

**insert** **overwrite local directory** '/home/hadoop/apps/hive\_test\_data/export\_data'

row format delimited **fields terminated by '\t'** select \* from student\_info;

导出数据到本地的常用方法

hive -e"select \* from rel.student\_info"> ./student\_info\_data.txt

默认结果分隔符：'\t'

**使用export导出表结构+数据**

**EXPORT TABLE** table\_name **TO** hdfs\_path';

## join关联

join查询时水平连接的，而union是水平连接的

table a 和 table b

**join或inner join**

两个表通过id关联，只把id值相等的数据查询出来。join的查询结果与inner join的查询结果相同。

select \* from a **join** b **on** a.id=b.id;

等同于 select \* from a **inner join** b **on** a.id=b.id;

**full outer join或full join**

两个表通过id关联，把两个表的数据全部查询出来

select \* from a **full join** b on a.id=b.id;

**left join**

左连接时，左表中出现的join字段都保留，右表没有连接上的都为空

select \* from a **left join** b **on** a.id=b.id;

**right join**

右连接时，右表中出现的join字段都保留，左表没有连接上的都是空

select \* from a right **join** b **on** a.id=b.id;

**left semi join**

左半连接实现了类似IN/EXISTS的查询语义，输出符合条件的左表内容。

hive不支持in …exists这种关系型数据库中的子查询结构，hive暂时不支持右半连接。

例如：select a.id, a.name from a where a.id in (select b.id from b);

使用Hive对应于如下语句：

select a.id,a.name from a left semi join b on a.id = b.id;

**特殊的JOIN – MAP JOIN**

使用分布式缓存将小表数据加载都各个map任务中，在map端完成join，map任务输出后，不需要将数据拷贝到reducer阶段再进行join，降低的数据在网络节点之间传输的开销。多表关联数据倾斜优化的一种手段。多表连接，如果只有一个表比较大，其他表都很小，则join操作会转换成一个只包含map的Job。运行日志中会出现Number of reduce tasks is set to 0 since there's no reduce operator没有reduce的提示。例如：select /\*+ mapjoin(b) \*/ a.id, a.name from a join b on a.id = b.id

默认情况下，hive.auto.convert.join属性设置为true，表示已经开启了优化。

hive内置函数

创建用户评分表

create table rel.user\_core\_info(

user\_id string,

age int,

gender string,

core int

)

row format delimited fields terminated by '\t'

lines terminated by '\n'

stored as textfile;

load data local inpath '/home/hadoop/apps/hive\_test\_data/user\_core.txt' into table rel.user\_core\_info;

1. 条件函数 case when

语法1：CASE a WHEN b THEN c [WHEN d THEN e]\* [ELSE f] END

说明：如果a等于b，那么返回c；如果a等于d，那么返回e；否则返回f

例如：

hive> select case 1 when 2 then 'two' when 1 then 'one' else 'zero' end;

one

语法2：CASE WHEN a THEN b [WHEN c THEN d]\* [ELSE e] END

说明：如果a为TRUE，则返回b；如果c为TRUE，则返回d；否则返回e

例如：

hive> select case when 1=2 then 'two' when 1=1 then 'one' else 'zero' end;

one

查询用户评分表，每个年龄段的最大评分值

select gender,

case when age<=20 then 'p0' when age>20 and age<=50 then 'p1' when age>=50 then 'p3' else 'p0' end,

max(core) max\_core

from rel.user\_core\_info

group by gender,

case when age<=20 then 'p0' when age>20 and age<=50 then 'p1' when age>=50 then 'p3' else 'p0' end;

2. 自定义UDF函数

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）。

UDF 作用于单个数据行，产生一个数据行作为输出。

步骤：

1. 先开发一个java类，继承UDF，并重载evaluate方法

2. 打成jar包上传到服务器

3. 在使用的时候将jar包添加到hive的classpath

hive>add jar /home/hadoop/apps/hive\_test\_data/HiveUdfPro-1.0-SNAPSHOT.jar;

4. 创建临时函数与开发好的java class关联

hive>create temporary function age\_partition as 'cn.chinahadoop.udf.AgePartitionFunction';

5. 即可在hql中使用自定义的函数

select gender,

age\_partition(age),

max(core) max\_core

from rel.user\_core\_info

group by gender,

age\_partition(age);

## Hiveserver2

作用：

1为hive提供一种允许客户端远程访问的服务

2 基于thrift协议，支持跨平台、跨语言对hive访问

### 1. HiveServer1

HiveServer是一种可选服务，允许远程客户端可以使用各种编程语言向Hive提交请求并检索结果。HiveServer是建立在Apache ThriftTM（[http://thrift.apache.org/）](http://thrift.apache.org/%EF%BC%89) 之上的，因此有时会被称为Thrift Server，这可能会导致混乱，因为新服务HiveServer2也是建立在Thrift之上的．自从引入HiveServer2后，HiveServer也被称为HiveServer1。

警告

HiveServer无法处理来自多个客户端的并发请求.这实际上是HiveServer导出的Thrift接口所施加的限制，也不能通过修改HiveServer源代码来解决。

HiveServer2对HiveServer进行了重写，来解决这些问题，从Hive 0.11.0版本开始。建议使用HiveServer2。

从Hive1.0.0版本（以前称为0.14.1版本）开始，HiveServer开始被删除。请切换到HiveServer2。

**通过10000端口远程访问hadoop上的hive**

1.启动hiveserver2服务器，监听端口10000

$>hive --service hiveserver2 &

Beeline与hive cli类似，是连接hiveserver2的客户端。

2.通过beeline命令行连接到hiveserver2

$>beeline //进入beeline命令行(于hive --service beeline)

$beeline>!help //查看帮助

$beeline>!quit //退出

**$beeline>!connect jdbc:hive2://localhost:10000/mydb2 //连接到hibve数据**

$beeline>show databases ;

$beeline>use mydb2 ;

$beeline>show tables; //显式表

## Hive的事务

Since Hive version 0.13.0, Hive fully supports row-level transactions by offering full **Atomicity, Consistency, Isolation, and Durability** (**ACID**) to Hive. For now, all the transactions are autocommuted and only support data in the **Optimized Row Columnar** (**ORC**) file (available since Hive 0.11.0) format and in bucketed tables

**开启事务**

1、hive的DML操作

在执行update、delete操作的时候需要开启事务的支持

2、hive事务的开启需要以下的依赖的条件

只支持ORCFile文件格式

默认关闭，需要手动开启

表必须支持分桶

0.14开始支持insert，delete，update操作

必须加入配置

3、需要修改的配置

<property>

<name>hive.support.concurrency</name>

<value>true</value>

</property>

<property>

<name>hive.enforce.bucketing</name>

<value>true</value>

</property>

<property>

<name>hive.exec.dynamic.partition.mode</name>

<value>nonstrict</value>

</property>

<property>

<name>hive.txn.manager</name>

<value>org.apache.hadoop.hive.ql.lockmgr.DbTxnManager</value>

</property>

<property>

<name>hive.compactor.initiator.on</name>

<value>true</value>

</property>

<property>

<name>hive.compactor.worker.threads</name>

<value>1</value>

</property>

**建表语句**

CREATE TABLE tx(id int,name string,age int) CLUSTERED BY (id) INTO 3 BUCKETS ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' stored as **orc** TBLPROPERTIES (**'transactional'='true'**);

## 聚合函数

count

group by

distinct

## 调优

### 查询计划-EXPLAIN

Hive使用explain命令查看一个查询计划，而并不会运行该查询。

它会列出整个查询阶段的各个操作，一般用于分析。

**EXPLAIN [EXTENDED|DEPENDENCY|AUTHORIZATION] hive\_query**

### limit优化测，避免全部查询.

set hive.limit.optimize.enable=true

### 本地模式

(用于本地测试)

set mapred.job.tracker=local;

set hive.exec.mode.local.auto=true //自动本地模式,测试

### 设置MR的数量

set hive.exec.reducers.bytes.per.reducer=750000000;

//设置reduce处理的字节数。

### 严格模式,

$hive>set hive.mapred.mode=strict

//1.分区表必须指定分区进行查询

//2.order by时必须使用limit子句。

//3.不允许笛卡尔积.

## 函数

HIVE中支持三种函数：

标准函数：如abs()(求绝对值),ucase()(转换大写),reverse()(字符串反转)等

聚合函数：count(),sum(),avg()

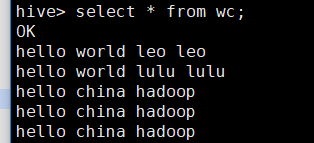
表生成函数: explode(),split(),array 接收0个或多个输入，产生多列或多行

**查看函数的详细信息**

desc function function\_name;

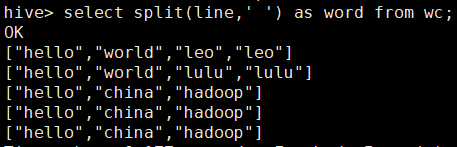
desc function extended function\_name;

表wc中，以空格分割每行数据，表中只有一个字段line,string类型



* 函数split(string,pattern)：按照分割符分割字符串成数组

例如: select split(line,' ') as word from wc;

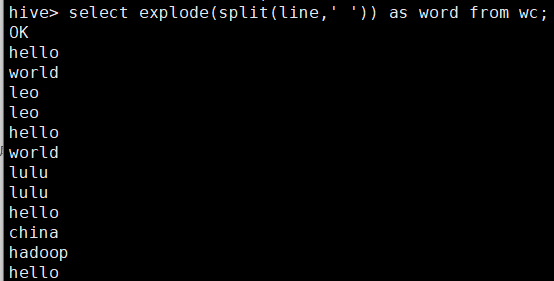


把line字段作为函数的输入，split会对每行进行分隔，变成数组。

* 函数explode()：separates the elements of array a into multiple rows, or the elements of a map into multiple rows and columns。把数组的元素分成多行。

//把上面输出的每行分割成每个单词

select explode(split(line,' ')) as word from wc;



词频统计

select t.word,count(t.word) as c from

(select explode(split(line,' ')) as word from wc) as t

group by t.word order by c desc;

### 自定义函数(USER DEFINE FUNCTION - UDF)

步骤

1.继承UDF类 ，实现evaluate()方法，此方法可以重载。@Desciption注解可以用于为自定义的函数添加描述信息。返回值只能是Hive中可以序列化的数据类型

@Description(name = "myAdd",  
 value = "\_FUNC\_(a,b) - from the input integer a,b "+  
 "returns the result of a+b",  
 extended = "Example:\n"  
 + " > SELECT \_FUNC\_(a,b); the result is a+b\n"  
 + " > SELECT \_FUNC\_(a,b,c) the result is a+b+c;")

2.添加到hive的类路径(

可以直接手动添加到hive的lib目录

也可以在hive中使用命令添加：$hive> ADD JAR jar\_full\_path

3.添加临时函数

CREATE TEMPERAY FUNCTION function\_name AS ‘jar包中的类路径’。

4.删除临时函数

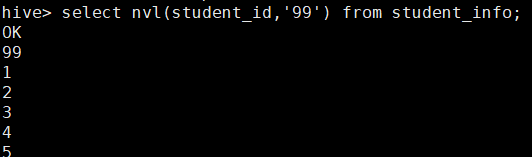
DROP TEMPERORY FUNCTION IF EXISTS function\_name;

以上方法创建的函数在一个hive的session中有效。

### Generic UDF

Generic UDF 可以更好的支持null值处理，同时可以处理一些标准的UDF无法支持的变成操作。这个nvl(value,defalut\_value)函数返回的是value和default\_value中的第一个非空值。





当student\_id为空值是，将设置为99

## Hive Streaming

在前面我们看到了UDF、UDTF、UDAF的实现并不是很简单，而且还要求对Java比较熟悉，而Hive设计的初衷是方便那些非Java人员使用。因此，Hive提供了另一种数据处理方式——Streaming，这样就可以不需要编写Java代码了，其实Streaming处理方式可以**支持很多语言**。但是，Streaming的执行效率通常比对应编写的UDF或改写InputFormat对象的方式要低。管道中序列化然后反序列化数据通常时低效的。而且以通常的方式很难调试整个程序。

Hive中提供了多种语法来使用Streaming，包括：

* MAP()
* REDUCE()
* TRANSFORM()

但是，注意MAP()实际上并非在Mapper阶段执行Streaming，正如REDUCE()实际上并非在Reducer阶段执行Streaming。因此，相同的功能，通常建议使用TRANSFORM()语句，这样可以避免产生疑惑。

### Streaming的编写和使用

Streaming的实现需要TRANSFORM()函数和USING关键字，TRANSFORM()的参数是表的列名，USING关键字用于指定脚本，脚本的作用是对查询的值进行相应的操作，如下例中 ‘/bin/cat’就是输出到linux的终端，用户可以使用其他语言编写任意的脚本进行操作。

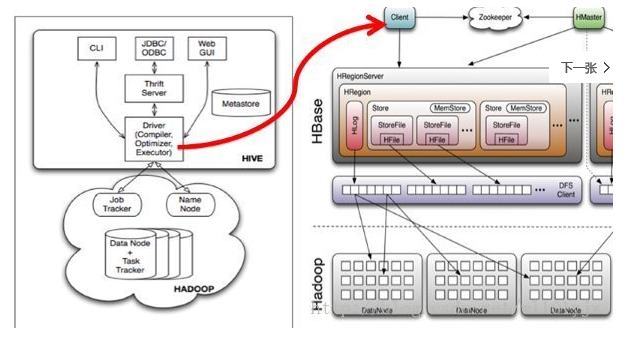
SELECT TRANSFORM(e.name, e.salary)

USING '/bin/cat' AS name, salary

FROM employee e;

## Hive整合HBase

Hive与HBase整合的实现是利用两者本身对外的API接口互相通信来完成的，其具体工作交由Hive的lib目录中的hive-hbase-handler-\*.jar工具类来实现，通信原理如下图所示。



Hive整合HBase后的使用场景：

（一）通过Hive把数据加载到HBase中，数据源可以是文件也可以是Hive中的表。

（二）通过整合，让HBase支持JOIN、GROUP等SQL查询语法。

（三）通过整合，不仅可完成HBase的数据实时查询，也可以使用Hive查询HBase中的数据完成复杂的数据分析。

Hive映射HBase表

配置方式略 CDH中在hive配置中选择整合hbase即可

hive中创建映射表

> create table hive\_hbase\_test(key int,value string) stored by '**org.apache.hadoop.hive.hbase.HBaseStorageHandler**' with

serdeproperties("hbase.columns.mapping"="**:key**,cf1:val") tblproperties("hbase.table.name"="hive\_hbase\_test");

备注：在hive中创建表hive\_hbase\_test，这个表包括两个字段（int型的key和string型的value），映射为hbase中的表hive\_hbase\_test，**key对应hbase的rowkey，value对应hbase的cf1:val列**。

然后在hbase中会看到刚创建的表。

创建一张新表poke表并加载测试数据，然后使用查询语句把数据加载到映射表中（不能直接把数据加载到映射表中）

从hive中导入数据后，hbase中会出现相同的数据。

需要说明以下几点：

1.Hive映射表的字段是HBase表字段的子集。整合之后的Hive表不能被修改。

2.Hive中的映射表不能直接插入数据，所以需要通过将数据加载到另一张poke表，然后通过查询poke表将数据加载到映射表。

3.上述示例是通过创建内部表的方式将Hive表映射到HBase表，HBase表会自动创建，而且Hive表被删除后HBase表也会自动删除。

4.如果HBase表已有数据，可以通过创建Hive外部表的方式将Hive表映射到HBase表，通过HQLHive表实现对HBase表的数据分析。Hive表删除将不会对HBase表造成影响。创建外部表的方法如下：

> create **external** table hive\_hbase\_test(key int,value string)stored by 'org.apache.hadoop.hive.hbase.HBaseStorageHandler' with

serdeproperties("hbase.columns.mapping"=":key,cf1:val") tblproperties("hbase.table.name"="hive\_hbase\_test");