Hive 基本操作

目录

学习目标：掌握 Hive 的各种基本操作：创建库和表，插入数据，查询数据

1、DDL 操作

1.1、库操作

1.1.1、创建库

语法结构：

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
  [COMMENT database_comment]
  [LOCATION hdfs_path]
  [WITH DBPROPERTIES (property_name=property_value, ...)];
```

创建库的使用方式：

1、 创建普通库

create database dbname;

2、 创建库的时候检查存与否

create databse if not exists dbname;

3、 创建库的时候带注释

create database if not exists dbname comment 'create my db named dbname';

4、 创建带属性的库

create database if not exists dbname with dbproperties ('a'='aaa','b'='bbb');

create database if not exists myhive with dbproperties ('a'='aaa','b'='bbb');

1.1.2、查看库

1、查看有哪些数据库

show databases;

2、显示数据库的详细属性信息

语法：**desc database [extended] dbname;**

示例：**desc database extended myhive;**

3、查看正在使用哪个库

select current_database();

4、查看创建库的详细语句

show create database mydb;

1.1.3、删除库

删除库操作：

drop database dbname;

drop database if exists dbname;

默认情况下，hive 不允许删除包含表的数据库，有两种解决办法：

1、 手动删除库下所有表，然后删除库

2、 使用 cascade 关键字

drop database if exists dbname cascade;

默认情况下就是 restrict

drop database if exists myhive ==== drop database if exists myhive restrict

1.1.4、切换库

切换库操作：

语法：use database_name

实例：**use myhive;**

1.2、表操作

1.2.1、创建表

1、 建表语句

语法结构：

```
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]
```

详情请参见：

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-CreateTable

2、 建表语句相关解释

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。

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），如果不存在，则会自动创建该目录。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。（经典面试问题）

外部表和内部表的选择：

1、 如果数据已经存储在 HDFS 上了，然后需要使用 Hive 去进行分析，并且该份数据还有可能要使用其他的计算引擎做计算之用，请使用外部表

2、 如果一份数据仅仅只是使用 Hive 做统计分析，那么可以使用内部表

不管使用内部表和外部表，**表的数据存储路径都是可以通过 location 指定的!!!!!!**

推荐方式：

1、 创建内部表的时候，最好别指定 location，就存储在默认的仓库路径

2、 如果要指定外部路径，那么请创建该表为外部表

PARTITIONED BY 在 Hive Select 查询中一般会扫描整个表内容，会消耗很多时间做没必要的工作。有时候只需要扫描表中关心的一部分数据，因此建表时引入 partition 概念。个

表可以拥有一个或者多个分区，每个分区以文件夹的形式单独存在表文件夹的目录下，分区是以字段的形式在表结构中存在，通过 desc table 命令可以查看到字段存在，但是该字段不存放实际的数据内容，仅仅是分区的表示。

分区建表分为 2 种：
一种是单分区，也就是说在表文件夹目录下只有一级文件夹目录
一种是多分区，表文件夹下出现多文件夹嵌套模式

LIKE：允许用户复制现有的表结构，但是不复制数据。
示例：create table tableA like tableB（创建一张 tableA 空表复制 tableB 的结构）

COMMENT：可以为表与字段增加描述

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, ...)]

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。

STORED AS TEXTFILE | SEQUENCEFILE | RCFILE
如果文件数据是纯文本，可以使用 STORED AS TEXTFILE，默认也是 textFile 格式，可以通过执行命令 **set hive.default.fileformat**，进行查看，如果数据需要压缩，使用 STORED AS SEQUENCEFILE。

A、**默认格式 TextFile**，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(系统自动检查，执行查询时自动解压)，但使用这种方式，hive 不会对数据进行切分，从而无法对数据进行并行操作

B、**SequenceFile** 是 Hadoop API 提供的一种**二进制文件**支持，文件内容是以序列化的 kv 对象来组织的，其具有使用方便、可分割、可压缩的特点。 SequenceFile 支持三种压缩选择：NONE，RECORD，BLOCK。Record 压缩率低，一般建议使用 BLOCK 压缩

C、**RCFILE 是一种行列存储相结合的存储方式**。首先，其将数据按行分块，保证同一个 record 在一个块上，避免读一个记录需要读取多个 block。其次，块数据列式存储，有利于数据压缩和快速的列存取。相比 TEXTFILE 和 SEQUENCEFILE，RCFILE 由于列式存储方式，数据加载时性能消耗较大，但是具有较好的压缩比和查询响应。数据仓库的特点是一次写入、多次读取，因此，整体来看，RCFILE 相比其余两种格式具有较明显的优势

CLUSTERED BY

对于每一个表（table）或者分区，Hive 可以进一步组织成桶，也就是说桶是更为细粒度的数据范围划分。Hive 也是针对某一列进行桶的组织。Hive 采用对列值哈希，然后除以桶的个数求余的方式决定该条记录存放在哪个桶当中。

把表（或者分区）组织成桶（Bucket）有两个理由：

（1）**获得更高的查询处理效率**。桶为表加上了额外的结构，Hive 在处理有些查询时能利用这个结构。具体而言，连接两个在（包含连接列的）相同列上划分了桶的表，可以使用 Map 端连接（Map-side join）高效的实现。比如 JOIN 操作。对于 JOIN 操作两个表有一个相同的列，如果对这两个表都进行了桶操作。那么将保存相同列值的桶进行 JOIN 操作就可以，可以大大较少 JOIN 的数据量。

（2）**使取样（sampling）更高效**。在处理大规模数据集时，在开发和修改查询的阶段，如果能在数据集的一小部分数据上试运行查询，会带来很多方便。

（经典面试题）

LOCATION：指定数据文件存放的 HDFS 目录，不管内部表还是外表，都可以指定。不指定就在默认的仓库路径。

3、 Hive 建表示例

```
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 31 BUCKETS
        ROW FORMAT DELIMITED
        FIELDS TERMINATED BY '\t'
        COLLECTION ITEMS TERMINATED BY '\t'
        MAP KEYS TERMINATED BY '\t'
        LINES TERMINATED BY '\n'
    STORED AS TEXTFILE
    LOCATION '/myhive'
```

执行命令查看表结构：hive> **desc formatted page_view;**

```
# col_name              data_type               comment
viewtime                int
userid                  bigint
page_url                string
referrer_url            string
ip                      string                  IP Address of the User

# Partition Information
# col_name              data_type               comment

dt                      string
country                 string
```

```
# Detailed Table Information
Database:               mytest
Owner:                  hadoop
CreateTime:             Tue Jan 10 20:15:15 CST 2017
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://hadoop02:9000/myhive
Table Type:             MANAGED_TABLE
Table Parameters:
        comment                 This is the page view table
        transient_lastDdlTime   1484050515

# Storage Information
SerDe Library:      org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe
InputFormat:        org.apache.hadoop.mapred.TextInputFormat
OutputFormat:       org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat
Compressed:         No
Num Buckets:        31
Bucket Columns:     [userid]
Sort Columns:       [Order(col:viewtime, order:1)]
Storage Desc Params:
        colelction.delim    \t
        field.delim         \t
        line.delim          \n
        mapkey.delim        \t
        serialization.format    \t
```

Hive 使用一个 Inputformat 对象将输入流分割成记录；使用一个 Outputformat 对象将记录格式化为输出流，使用序列化/反序列化器 SerDe 做记录的解析（记录和列的转换）。

它们的默认值分别是：

Inputformat：org.apache.hadoop.mapred.TextInputFormat

Outputformat：org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

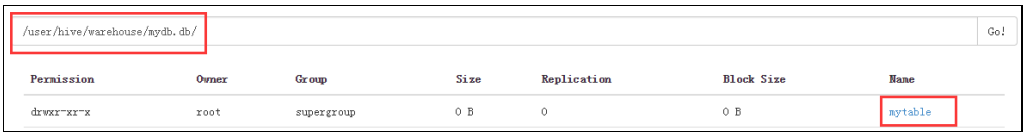
SerDe：org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe

4、 具体实例

a、 创建内部表

```
create table mytable (id int, name string)
row format delimited fields terminated by ',' stored as textfile;
```

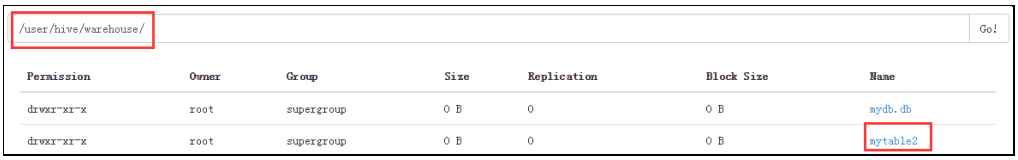
```
hive> create table mytable (id int, name string)
    > row format delimited fields terminated by ',' stored as textfile;
OK
Time taken: 0.11 seconds
hive> show tables;
OK
mytable
t_user1
t_user2
values__tmp__table__1
Time taken: 0.027 seconds, Fetched: 4 row(s)
hive>
```



b、 创建外部表

create external table mytable2 (id int, name string) row format delimited fields terminated by ',' location '/user/hive/warehouse/mytable2';

```
hive> Create external table mytable2 (id int, name string) row format delimited fields terminated by ',' location '/user/
hive/warehouse/mytable2';
OK
Time taken: 0.51 seconds
hive> show tables;
OK
mingxing
mingxing2
mingxing4_ptn
mingxing6
mingxing_ptn
mystu
mystu_buck
mytable2
stu
stu_buck
student_ext
student_ptn
sut_buck
Time taken: 0.088 seconds, Fetched: 13 row(s)
hive>
```



c、 创建分区表

create table mytable3(id int, name string) partitioned by(sex string)

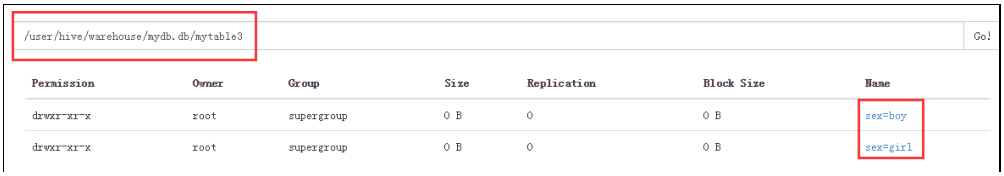
row format delimited fields terminated by ',' stored as textfile;

```
hive> create table mytable3(id int, name string)
    > partitioned by(sex string) row format delimited fields terminated by ','stored as textfile;
OK
Time taken: 0.1 seconds
hive> show tables;
OK
mytable
mytable2
mytable3
t_user1
t_user2
values__tmp__table__1
Time taken: 0.036 seconds, Fetched: 6 row(s)
hive>
```

插入数据

插入男分区数据：load data local inpath '/root/hivedata/mingxing.txt' overwrite into table mytable3 partition(sex='boy');

插入女分区数据：load data local inpath '/root/hivedata/mingxing.txt' overwrite into table mytable3 partition(sex='girl');




```
0: jdbc:hive2://hadoop02:10000> select * from mytable3;
+--------------+----------------+----------------+--+
| mytable3.id  | mytable3.name  | mytable3.sex   |
+--------------+----------------+----------------+--+
| 1            | huangbo        | boy            |
| 2            | dengchao       | boy            |
| 3            | xuzheng        | boy            |
| 4            | wangbaoqiang   | boy            |
| 1            | huangbo        | girl           |
| 2            | dengchao       | girl           |
| 3            | xuzheng        | girl           |
| 4            | wangbaoqiang   | girl           |
+--------------+----------------+----------------+--+
8 rows selected (0.216 seconds)
0: jdbc:hive2://hadoop02:10000> select * from mytable3 where sex='boy';
+--------------+----------------+----------------+--+
| mytable3.id  | mytable3.name  | mytable3.sex   |
+--------------+----------------+----------------+--+
| 1            | huangbo        | boy            |
| 2            | dengchao       | boy            |
| 3            | xuzheng        | boy            |
| 4            | wangbaoqiang   | boy            |
+--------------+----------------+----------------+--+
4 rows selected (0.422 seconds)
0: jdbc:hive2://hadoop02:10000>
```

查询表分区：

show partitions mytable3

```
0: jdbc:hive2://hadoop02:10000> show partition mytable3;
Error: Error while compiling statement: FAILED: ParseExceptio
ytable3' in ddl statement (state=42000,code=40000)
0: jdbc:hive2://hadoop02:10000> show partitions mytable3;
+------------+--+
| partition  |
+------------+--+
| sex=boy    |
| sex=girl   |
+------------+--+
2 rows selected (0.157 seconds)
0: jdbc:hive2://hadoop02:10000>
```

d、 创建分桶表

create table stu_buck(Sno int,Sname string,Sex string,Sage int,Sdept string)
clustered by(Sno) sorted by(Sno DESC) into 4 buckets
row format delimited fields terminated by ',';

```
44 rows selected (0.526 seconds)
0: jdbc:hive2://hadoop01:10000> create table stu_buck(Sno int,Sname string,Sex string,Sage int,Sdept string)
0: jdbc:hive2://hadoop01:10000> clustered by(Sno) sorted by(Sno DESC) into 4 buckets
0: jdbc:hive2://hadoop01:10000> row format delimited fields terminated by ',';
No rows affected (0.236 seconds)
0: jdbc:hive2://hadoop01:10000> show tables;
+--------------+--+
|   tab_name   |
+--------------+--+
| stu_buck     |
| student      |
| student1     |
| student_ext  |
| student_ptn  |
+--------------+--+
```

e、 使用 like 关键字拷贝表

// 不管老表是内部表还是外部表，new_table 都是内部表

create table new_table like mytable;

// 不管老表是内部表还是外部表，new_table 都是外部表，如果加 external 关键字

create external table if not exists new_table like mytable;

1.2.2、修改表

1、重命名表

语法结构：ALTER TABLE table_name **RENAME TO** new_table_name

示例：

```
hive> alter table student rename to studentss;
OK
Time taken: 0.16 seconds
hive> select * from studentss;
OK
95001   李勇    男      20      CS
95011   包小柏  男      18      MA
95009   梦圆圆  女      18      MA
```

2、修改表属性

语法结构：

```
ALTER TABLE table_name SET TBLPROPERTIES table_properties;
table_properties: (property_name = property_value, property_name = property_value, ... )
```

实例：

ALTER TABLE table_name SET TBLPROPERTIES ('comment' = 'my new students table');

3、修改 SerDe 信息

语法结构：

```
ALTER TABLE table_name [PARTITION partition_spec] SET SERDE serde_class_name [WITH SERDEPROPERTIES serde_properties];

ALTER TABLE table_name [PARTITION partition_spec] SET SERDEPROPERTIES serde_properties;

serde_properties:
   : (property_name = property_value, property_name = property_value, ... )
```

实例：

更改列分隔符：ALTER TABLE student SET SERDEPROPERTIES ('field.delim' = '-');

4、增加/删除/改变/替换列

语法结构：

```
ALTER TABLE name ADD COLUMNS (col_spec[, col_spec ...])
ALTER TABLE name CHANGE c_name new_name new_type [FIRST|AFTER c_name]
ALTER TABLE name REPLACE COLUMNS (col_spec[, col_spec ...])
```

（注意：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。）

ADD 示例：

```
hive> desc studentss;
OK
sno                     int
sname                   string
sex                     string
sage                    int
sdept                   string
Time taken: 0.124 seconds, Fetched: 5 row(s)
hive> alter table studentss add columns (course string);
OK
Time taken: 0.154 seconds
hive> desc studentss;
OK
sno                     int
sname                   string
sex                     string
sage                    int
sdept                   string
course                  string
Time taken: 0.108 seconds, Fetched: 6 row(s)
hive>
```

```
hive> desc studentss;
OK
id                      int
name                    string
address                 string
Time taken: 0.105 seconds, Fetched: 3 row(s)
hive> alter table studentss change column id ids int;
OK
Time taken: 0.165 seconds
hive> desc studentss;
OK
ids                     int
name                    string
address                 string
Time taken: 0.1 seconds, Fetched: 3 row(s)
hive>
```

```
hive> desc studentss;
OK
sno                     int
sname                   string
sex                     string
sage                    int
sdept                   string
course                  string
Time taken: 0.108 seconds, Fetched: 6 row(s)
hive> alter table studentss replace columns (id int, name string, address string);
OK
Time taken: 0.134 seconds
hive> desc studentss;
OK
id                      int
name                    string
address                 string
Time taken: 0.109 seconds, Fetched: 3 row(s)
hive>
```

5、增加/删除分区

增加分区语法结构：

```
ALTER TABLE table_name ADD [IF NOT EXISTS] partition_spec [ LOCATION 'location1' ]
partition_spec [ LOCATION 'location2' ] ...

partition_spec
: PARTITION (partition_col1 = partition_col_value1, partition_col2 = partiton_col_value2, ...)
```

删除分区语法结构：

```
ALTER TABLE table_name DROP partition_spec, partition_spec, ...
```

添加分区示例：

// 不指定分区路径，默认路径

ALTER TABLE student_p **ADD** partition(part='a') partition(part='b');

// 指定分区路径

ALTER TABLE student_p **ADD** IF NOT EXISTS

partition(part='bb') location '/myhive_bb' partition(part='cc') location '/myhive_cc';

```
hive> alter table student add partition(stat_date='20140101') location '/user/hive/warehouse/student' partition(stat_date='20140102')
;
OK
Time taken: 1.669 seconds
hive> alter table student add partition(stat_date='20140103');
OK
Time taken: 1.128 seconds
hive> dfs -ls /user/hive/warehouse/student;
Found 4 items
drwxr-xr-x   - root supergroup          0 2013-12-30 13:25 /user/hive/warehouse/student/stat_date=20131230
drwxr-xr-x   - root supergroup          0 2013-12-30 17:06 /user/hive/warehouse/student/stat_date=20131231
drwxr-xr-x   - root supergroup          0 2013-12-30 17:09 /user/hive/warehouse/student/stat_date=20140102
drwxr-xr-x   - root supergroup          0 2013-12-30 17:10 /user/hive/warehouse/student/stat_date=20140103
hive>
```

```
hive> show partitions student;
OK
stat_date=20131230
stat_date=20131231
stat_date=20140101
stat_date=20140102
stat_date=20140103
Time taken: 0.878 seconds, Fetched: 5 row(s)
hive> alter table student drop partition(stat_date='20131231');
Dropping the partition stat_date=20131231
OK
Time taken: 4.227 seconds
hive> alter table student drop partition(stat_date='20140101'),partition(stat_date='20140102');
Dropping the partition stat_date=20140101
Dropping the partition stat_date=20140102
OK
Time taken: 2.131 seconds
hive>
```

// 修改分区路径示例

ALTER TABLE student_p partition (part='bb') **SET** location '/myhive_bbbbb';

// 删除分区示例

ALTER TABLE student_p **DROP** if exists partition(part='aa');

ALTER TABLE student_p **DROP** if exists partition(part='aa') if exists partition(part='bb');

最后补充：

1、 防止分区被删除：alter table student_p partition (part='aa') **enable no_drop**;

2、 防止分区被查询：alter table student_p partition (part='aa') **enable offline**;

enable 和 disable 是反向操作

1.2.3、删除表

语法结构：

```
DROP TABLE [IF EXISTS] table_name;
```

命令：**drop table if exists mytable;**

```
hive> show tables;
OK
mytable
mytable2
mytable3
stu_buck
studentss
t_user1
t_user2
values__tmp__table__1
Time taken: 0.036 seconds, Fetched: 8 row(s)
hive> drop table if exists mytable;
OK
Time taken: 0.143 seconds
hive> show tables;
OK
mytable2
mytable3
stu_buck
studentss
t_user1
t_user2
values__tmp__table__1
Time taken: 0.042 seconds, Fetched: 7 row(s)
hive>
```

1.2.4、清空表

语法：

```
TRUNCATE TABLE table_name [PARTITION partition_spec];
```

实例：

truncate table student;

truncate table student_ptn partition(city='beijing');

1.3、其他辅助命令

show databases; show databases like 'my*';	查看数据库列表
show tables; show tables in db_name;	查看数据表
show create table table_name;	查看数据表的建表语句
show functions;	查看 hive 函数列表
show partitions table_name; show partitions table_name partition(city='beijing')	查看 hive 表的分区
desc table_name; desc extended table_name; desc formatted table_name;	查看表的详细信息（元数据信息）
desc database db_name; desc database extended db_name;	查看数据库的详细属性信息
truncate table table_name;	清空数据表

2、DML 操作

2.1、Load 装载数据

语法结构：

LOAD DATA [LOCAL] INPATH 'filepath' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]

说明：

1、 LOAD 操作只是单纯的复制或者移动操作，将数据文件移动到 Hive 表对应的位置。

2、 LOCAL 关键字

如果指定了 LOCAL， LOAD 命令会去查找本地文件系统中的 filepath。

如果没有指定 LOCAL 关键字，则根据 inpath 中的 uri 查找文件

注意：uri 是指 hdfs 上的路径，分简单模式和完整模式两种，例如：

简单模式：/user/hive/project/data1

完整模式：hdfs://namenode_host:9000/user/hive/project/data1

3、 filepath：

相对路径，例如：project/data1

绝对路径，例如：/user/home/project/data1

包含模式的完整 URI，列如：hdfs://namenode_host:9000/user/home/project/data1

注意：inpath 子句中的文件路径下，不能再有文件夹。

4、 overwrite 关键字

如果使用了 OVERWRITE 关键字，则目标表（或者分区）中的内容会被删除，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。

具体实例：

1、 加载本地相对路径数据

```
hive> load data local inpath 'buckets.txt' into table student partition(stat_date='20131231');
Copying data from file:/root/app/datafile/buckets.txt
Copying file: file:/root/app/datafile/buckets.txt
Loading data to table default.student partition (stat_date=20131231)
Partition default.student{stat_date=20131231} stats: [num_files: 1, num_rows: 0, total_size: 77, raw_data_size: 0]
Table default.student stats: [num_partitions: 2, num_files: 2, num_rows: 0, total_size: 154, raw_data_size: 0]
OK
Time taken: 2.753 seconds
hive> dfs -ls /user/hive/warehouse/student/ds=20131231;
ls: `/user/hive/warehouse/student/ds=20131231': No such file or directory
Command failed with exit code = 1
Query returned non-zero code: 1, cause: null
hive> dfs -ls /user/hive/warehouse/student/stat_date=20131231
    > ;
Found 1 items
-rw-r--r--   2 root supergroup         77 2013-12-31 13:38 /user/hive/warehouse/student/stat_date=20131231/buckets.txt
hive>
```

2、 加载绝对路径数据

```
hive> load data local inpath '/root/app/datafile/buckets.txt' into table student partition(stat_date='20131230');
Copying data from file:/root/app/datafile/buckets.txt
Copying file: file:/root/app/datafile/buckets.txt
Loading data to table default.student partition (stat_date=20131230)
Partition default.student{stat_date=20131230} stats: [num_files: 1, num_rows: 0, total_size: 77, raw_data_size: 0]
Table default.student stats: [num_partitions: 2, num_files: 2, num_rows: 0, total_size: 154, raw_data_size: 0]
OK
Time taken: 2.536 seconds
hive> dfs -ls /user/hive/warehouse/student/stat_date=20131230;
Found 1 items
-rw-r--r--   2 root supergroup         77 2013-12-31 13:42 /user/hive/warehouse/student/stat_date=20131230/buckets.txt
hive>
```

3、 加载包含模式数据

```
hive> load data local inpath 'hdfs://192.168.11.191:9000/user/hive/warehouse/student/stat_date=20131230/buckets.txt'
    > into table student partition(stat_date='20131229');
FAILED: SemanticException [Error 10028]: Line 1:23 Path is not legal ''hdfs://192.168.11.191:9000/user/hive/warehouse/student/stat_dat
e=20131230/buckets.txt'': Source file system should be "file" if "local" is specified
hive> load data inpath 'hdfs://192.168.11.191:9000/user/hive/warehouse/student/stat_date=20131230/buckets.txt'
    > into table student partition(stat_date='20131229');
Loading data to table default.student partition (stat_date=20131229)
Partition default.student{stat_date=20131229} stats: [num_files: 1, num_rows: 0, total_size: 77, raw_data_size: 0]
Table default.student stats: [num_partitions: 3, num_files: 3, num_rows: 0, total_size: 231, raw_data_size: 0]
OK
Time taken: 2.726 seconds
hive> dfs -ls /user/hive/warehouse/student/stat_date=20131229;
Found 1 items
-rw-r--r--   2 root supergroup         77 2013-12-31 13:42 /user/hive/warehouse/student/stat_date=20131229/buckets.txt
hive>
```

4、 overwrite 关键字使用

```
hive> load data local inpath 'buckets.txt' overwrite into table student partition(stat_date='20131229');
Copying data from file:/root/app/datafile/buckets.txt
Copying file: file:/root/app/datafile/buckets.txt
Loading data to table default.student partition (stat_date=20131229)
Partition default.student{stat_date=20131229} stats: [num_files: 1, num_rows: 0, total_size: 77, raw_data_size: 0]
Table default.student stats: [num_partitions: 3, num_files: 4, num_rows: 0, total_size: 308, raw_data_size: 0]
OK
Time taken: 2.328 seconds
hive> load data local inpath 'buckets.txt' overwrite into table student;
FAILED: SemanticException [Error 10062]: Need to specify partition columns because the destination table is partitioned
hive>
```

2.2、Insert 插入数据

语法结构：

1、插入一条数据：

INSERT INTO TABLE table_name VALUES(XX,YY,ZZ);

2、利用查询语句将结果导入新表：

INSERT OVERWRITE [INTO] TABLE table_name [PARTITION (partcol1=val1, partcol2=val2 ...)] select_statement1 FROM from_statement

3、多重插入

FROM from_statement

INSERT OVERWRITE TABLE table_name1 [PARTITION (partcol1=val1, partcol2=val2 ...)] select_statement1

INSERT OVERWRITE TABLE table_name2 [PARTITION (partcol1=val1, partcol2=val2 ...)] select_statement2] ...

示例：

from mingxing

insert into table mingxing2 select id,name,sex,age

insert into table mingxing select id,name,sex ,age,department ;

从 mingxing 表中，按不同的字段进行查询得的结果分别插入不同的 hive 表

from student

```
insert into table ptn_student partition(city='MA') select id,name,sex,age,department where department='MA'
insert into table ptn_student partition(city='IS') select id,name,sex,age,department where department='IS';
insert into table ptn_student partition(city='CS') select id,name,sex,age,department where department='CS';
```

4、分区插入

分区插入有两种，一种是静态分区，另一种是动态分区。如果混合使用静态分区和动态分区，则静态分区必须出现在动态分区之前。现分别介绍这两种分区插入。

静态分区：

A)、创建静态分区表

B)、从查询结果中导入数据

C)、查看插入结果

动态分区：

静态分区需要创建非常多的分区，那么用户就需要写非常多的 SQL！Hive 提供了一个动态分区功能，其可以基于查询参数推断出需要创建的分区名称。

A)、创建分区表，和创建静态分区表是一样的

B)、参数设置

```
hive> set hive.exec.dynamic.partition=true;
hive> set hive.exec.dynamic.partition.mode=nonstrict;
```

注意：动态分区默认情况下是开启的。但是却以默认是"strict"模式执行的，在这种模式下要求至少有一列分区字段是静态的。这有助于阻止因设计错误导致查询产生大量的分区。但是此处我们不需要静态分区字段，估将其设为 nonstrict。

对应还有一些参数可设置：

```
set hive.exec.max.dynamic.partitions.pernode=100;      //每个节点生成动态分区最大个数
set  hive.exec.max.dynamic.partitions=1000;            //生成动态分区最大个数，如果自动分区数大于这个参数，将会报错
set hive.exec.max.created.files=100000;      //一个任务最多可以创建的文件数目
set dfs.datanode.max.xcievers=4096;          //限定一次最多打开的文件数
set hive.error.on.empty.partition=false;     //表示当有空分区产生时，是否抛出异常
```

小技能补充：如果以上参数被更改过，想还原，请使用 reset 命令执行一次即可

C)、动态数据插入

```
// 一个分区字段：
insert into table test2 partition (age) select name,address,school,age from students;

// 多个分区字段：
insert into table student_ptn2 partition(city='sa',zipcode) select id, name, sex, age, department, department as zipcode from studentss;
```

注意：查询语句 select 查询出来的动态分区 age 和 zipcode 必须放最后，和分区字段对应，不然结果会出错

D)、查看插入结果

select * from student_ptn2 where city='sa' and zipcode='MA';

5、CTAS（create table … as select …）

在实际情况中，表的输出结果可能太多，不适于显示在控制台上，这时候，将 Hive 的查询输出结果直接存在一个新的表中是非常方便的，我们称这种情况为 CTAS

展示：CREATE TABLE mytest AS SELECT name, age FROM test;

注意：CTAS 操作是原子的，因此如果 select 查询由于某种原因而失败，新表是不会创建的！

2.3、Insert 导出数据

语法结构：

单模式导出：

INSERT OVERWRITE [LOCAL] DIRECTORY directory1 select_statement

多模式导出：

FROM from_statement

INSERT OVERWRITE [LOCAL] DIRECTORY directory1 select_statement1

[INSERT OVERWRITE [LOCAL] DIRECTORY directory2 select_statement2] ...

具体实例：

1、 导出数据到本地：

insert overwrite local directory '/root/hivedata/student.txt' select * from studentss;

```
hive> insert overwrite local directory '/root/app/datafile/student1'
    > select * from student1;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1388055845553_0020, Tracking URL = http://cloud001:8088/proxy/application_1388055845553_
Kill Command = /root/app/hadoop-2.2.0/bin/hadoop job  -kill job_1388055845553_0020
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2013-12-31 15:29:48,667 Stage-1 map = 0%,  reduce = 0%
2013-12-31 15:29:59,168 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.04 sec
2013-12-31 15:30:00,463 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.04 sec
2013-12-31 15:30:01,817 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.04 sec
MapReduce Total cumulative CPU time: 1 seconds 40 msec
Ended Job = job_1388055845553_0020
Copying data to local directory /root/app/datafile/student1
Copying data to local directory /root/app/datafile/student1
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 1.04 sec   HDFS Read: 309 HDFS Write: 135 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 40 msec
OK
Time taken: 30.017 seconds
hive>
```

注意：数据写入到文件系统时进行文本序列化，且每列用^A 来区分，\n 为换行符。用 more 命令查看时不容易看出分割符，可以使用: sed -e 's/\x01/\t/g' filename 来查看。

2、 导出数据到 HDFS

insert overwrite directory '/student' select * from studentss where age >= 20;

insert overwrite directory 'hdfs://hadoop02:9000/user/hive/warehouse/mystudent' select * from studentss;

```
hive> insert overwrite directory 'hdfs://192.168.11.191:9000/user/hive/warehouse/mystudent'
    > select * from student1;
Total MapReduce jobs = 3
Launching Job 1 out of 3
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1388055845553_0025, Tracking URL = http://cloud001:8088/proxy/application_1388055845553_0025/
Kill Command = /root/app/hadoop-2.2.0/bin/hadoop job  -kill job_1388055845553_0025
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2013-12-31 15:51:09,328 Stage-1 map = 0%,  reduce = 0%
2013-12-31 15:51:18,386 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.89 sec
2013-12-31 15:51:19,552 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.89 sec
MapReduce Total cumulative CPU time: 1 seconds 890 msec
Ended Job = job_1388055845553_0025
Stage-3 is selected by condition resolver.
Stage-2 is filtered out by condition resolver.
Stage-4 is filtered out by condition resolver.
Moving data to: hdfs://192.168.11.191:9000/tmp/hive-root/hive_2013-12-31_15-50-54_794_7628882137640941193-1/-ext-10000
Moving data to: hdfs://192.168.11.191:9000/user/hive/warehouse/mystudent
MapReduce Jobs Launched:
Job 0: Map: 1   Cumulative CPU: 1.89 sec   HDFS Read: 309 HDFS Write: 135 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 890 msec
OK
Time taken: 25.066 seconds
hive> dfs -ls /user/hive/warehouse/mystudent;
Found 1 items
-rw-r--r--   2 root supergroup        135 2013-12-31 15:51 /user/hive/warehouse/mystudent/000000_0
hive>
```

2.4、Select 查询数据

Hive 中的 SELECT 基础语法和标准 SQL 语法基本一致，支持 WHERE、DISTINCT、GROUP BY、ORDER BY、HAVING、LIMIT、子查询等；

1、select * from db.table1
2、select count(distinct uid) from db.table1
3、支持 select、union all、join（left、right、full join）、like、where、having、各种聚合函数、支持 json 解析
4、UDF（User Defined Function）/ UDAF/UDTF
5、不支持 update 和 delete
6、hive 虽然支持 in/exists（老版本是不支持的），但是 hive 推荐使用 semi join 的方式来代替实现，而且效率更高。
7、支持 case … when …

语法结构：

```
SELECT [ALL | DISTINCT] select_ condition, select_ condition, ...
FROM table_name a
[JOIN table_other b ON a.id = b.id]
[WHERE where_condition]
[GROUP BY col_list [HAVING condition]]
[CLUSTER BY col_list | [DISTRIBUTE BY col_list] [SORT BY col_list | ORDER BY col_list] ]
[LIMIT number]
```

说明：

1、 select_ condition 查询字段
2、 table_name 表名
3、 **order by(字段) 全局排序**，因此只有一个 reducer，只有一个 reduce task 的结果，比如文

件名是 000000_0，会导致当输入规模较大时，需要较长的计算时间。

4、 **sort by(字段) 局部排序**，不是全局排序，其在数据进入 reducer 前完成排序。因此，如果用 sort by 进行排序，并且设置 mapred.reduce.tasks>1，则 sort by 只保证每个 reducer 的输出有序，不保证全局有序。

那万一，我要对我的所有处理结果进行一个综合排序，而且数据量又非常大，那么怎么解决？我们不适用 order by 进行全数据排序，我们适用 sort by 对数据进行局部排序，完了之后，再对所有的局部排序结果做一个归并排序

5、 **distribute by(字段)** 根据指定的字段将数据分到不同的 reducer，且分发算法是 hash 散列。

6、 **cluster by(字段)** 除了具有 Distribute by 的功能外，还会对该字段进行排序。

因此，如果分桶和 sort 字段是同一个时，此时，**cluster by = distribute by + sort by**

如果我们要分桶的字段和要排序的字段不一样，那么我们就不能使用 clustered by

分桶表的作用：最大的作用是用来提高 join 操作的效率；

（思考这个问题：select a.id,a.name,b.addr from a join b on a.id = b.id;如果 a 表和 b 表已经是分桶表，而且分桶的字段是 id 字段做这个 join 操作时，还需要全表做笛卡尔积吗？）

具体实例：

1、 获取年龄大的三个学生

select id, age,name from student where stat_date= '20140101' order by age desc limit 3;

```
hive> select id,age,name from student where stat_date='20140101' order by age desc limit 3;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_1388055845553_0034, Tracking URL = http://cloud001:8088/proxy/application_1388055845553_0034/
Kill Command = /root/app/hadoop-2.2.0/bin/hadoop job  -kill job_1388055845553_0034
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2013-12-31 17:40:31,194 Stage-1 map = 0%,  reduce = 0%
2013-12-31 17:40:35,695 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.88 sec
2013-12-31 17:40:37,211 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.88 sec
2013-12-31 17:40:38,951 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.88 sec
2013-12-31 17:40:40,357 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.88 sec
2013-12-31 17:40:41,799 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.88 sec
2013-12-31 17:40:43,076 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.88 sec
2013-12-31 17:40:44,312 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.88 sec
2013-12-31 17:40:45,932 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.96 sec
2013-12-31 17:40:47,492 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.96 sec
2013-12-31 17:40:49,146 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 1.96 sec
MapReduce Total cumulative CPU time: 1 seconds 960 msec
Ended Job = job_1388055845553_0034
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 1.96 sec   HDFS Read: 289 HDFS Write: 34 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 960 msec
OK
7       25      wp
6       23      symbian
5       22      android
Time taken: 39.528 seconds, Fetched: 3 row(s)
hive>
```

2、 查询学生年龄按降序排序

set mapred.reduce.tasks=4;

select id, age, name from student sort by age desc;

```
hive> set mapred.reduce.tasks=4;
hive> select id,age,name from student sort by age desc;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Defaulting to jobconf value of: 4
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_1388055845553_0051, Tracking URL = http://cloud001:8088/proxy/application_1388055845553_0051/
Kill Command = /root/app/hadoop-2.2.0/bin/hadoop job  -kill job_1388055845553_0051
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 4
2014-01-01 10:38:55,679 Stage-1 map = 0%,  reduce = 0%
2014-01-01 10:39:08,210 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:09,797 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:11,153 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:12,371 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:13,571 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:14,710 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:15,850 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:17,006 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:18,127 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:19,246 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:20,369 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:21,470 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.14 sec
2014-01-01 10:39:22,652 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 3.37 sec
2014-01-01 10:39:23,880 Stage-1 map = 100%,  reduce = 50%, Cumulative CPU 4.85 sec
2014-01-01 10:39:25,038 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.57 sec
2014-01-01 10:39:26,562 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 7.57 sec
MapReduce Total cumulative CPU time: 7 seconds 570 msec
Ended Job = job_1388055845553_0051
MapReduce Jobs Launched:
Job 0: Map: 2  Reduce: 4   Cumulative CPU: 7.57 sec   HDFS Read: 816 HDFS Write: 72 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 570 msec
OK
4       18      mac
6       23      symbian
2       21      ljz
5       22      android
1       20      zxm
3       19      cds
7       25      wp
```

select id, age, name from student order by age desc;

```
hive> select id,age,name from student order by age desc;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_1388055845553_0052, Tracking URL = http://cloud001:8088/proxy/application_1388055845553_0052/
Kill Command = /root/app/hadoop-2.2.0/bin/hadoop job  -kill job_1388055845553_0052
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1
2014-01-01 10:42:35,660 Stage-1 map = 0%,  reduce = 0%
2014-01-01 10:42:50,180 Stage-1 map = 50%,  reduce = 0%, Cumulative CPU 1.31 sec
2014-01-01 10:42:51,821 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.65 sec
2014-01-01 10:42:53,475 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.65 sec
2014-01-01 10:42:54,899 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.65 sec
2014-01-01 10:42:56,129 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.65 sec
2014-01-01 10:42:57,311 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.65 sec
2014-01-01 10:42:59,270 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.65 sec
2014-01-01 10:43:00,830 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.65 sec
2014-01-01 10:43:02,153 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.25 sec
2014-01-01 10:43:03,402 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.25 sec
2014-01-01 10:43:04,990 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.25 sec
MapReduce Total cumulative CPU time: 4 seconds 250 msec
Ended Job = job_1388055845553_0052
MapReduce Jobs Launched:
Job 0: Map: 2  Reduce: 1   Cumulative CPU: 4.25 sec   HDFS Read: 816 HDFS Write: 72 SUCCESS
Total MapReduce CPU Time Spent: 4 seconds 250 msec
OK
7       25      wp
6       23      symbian
5       22      android
2       21      ljz
1       20      zxm
3       19      cds
4       18      mac
Time taken: 44.257 seconds, Fetched: 7 row(s)
hive>
```

select id, age, name from student distribute by age;

```
hive> select id,age,name from student distribute by age;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Defaulting to jobconf value of: 4
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_1388055845553_0053, Tracking URL = http://cloud001:8088/proxy/application_1388055845553_0053/
Kill Command = /root/app/hadoop-2.2.0/bin/hadoop job  -kill job_1388055845553_0053
Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 4
2014-01-01 10:45:28,885 Stage-1 map = 0%,  reduce = 0%
2014-01-01 10:45:44,005 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec
2014-01-01 10:45:45,724 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec
2014-01-01 10:45:47,156 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec
2014-01-01 10:45:48,373 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec
2014-01-01 10:45:49,509 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec
2014-01-01 10:45:51,005 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec
2014-01-01 10:45:52,504 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec
2014-01-01 10:45:53,750 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec
2014-01-01 10:45:54,920 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.16 sec
2014-01-01 10:45:56,079 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 3.4 sec
2014-01-01 10:45:57,861 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.85 sec
2014-01-01 10:45:59,543 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.85 sec
2014-01-01 10:46:01,234 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.85 sec
MapReduce Total cumulative CPU time: 6 seconds 850 msec
Ended Job = job_1388055845553_0053
MapReduce Jobs Launched:
Job 0: Map: 2  Reduce: 4   Cumulative CPU: 6.85 sec   HDFS Read: 816 HDFS Write: 72 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 850 msec
OK
1       20      zxm
7       25      wp
2       21      ljz
5       22      android
4       18      mac
6       23      symbian
3       19      cds
Time taken: 48.416 seconds, Fetched: 7 row(s)
hive>
```

这是分桶和排序的组合操作，对 id 进行分桶，对 age，id 进行降序排序

insert overwrite directory '/root/outputdata6' select * from mingxing2 distribute by id sort by age desc, id desc;

```
[root@hadoop01 ~]# hadoop fs -cat /root/outputdata6/000002_0
95006□孙庆□男□23□CS
95022□郑明□男□20□MA
95018□王一□女□19□IS
95014□王小丽□女□19□CS
95010□孔小涛□男□19□CS
95002□刘晨□女□19□IS
[root@hadoop01 ~]# hadoop fs -cat /root/outputdata6/000001_0
95013□冯伟□男□21□CS
95001□李勇□男□20□CS
95017□王风娟□女□18□IS
95009□梦圆圆□女□18□MA
95005□刘刚□男□18□MA
95021□周二□男□17□MA
[root@hadoop01 ~]#
```

这是分桶操作，按照 id 分桶，但是不进行排序

insert overwrite directory '/root/outputdata4' select * from mingxing2 distribute by id sort by age;

```
[root@hadoop01 ~]# hadoop fs -cat /root/outputdata4/000003_0
95015□王君□男□18□MA
95011□包小柏□男□18□MA
95019□邢小丽□女□19□IS
95007□易思玲□女□19□MA
95003□王敏□女□22□MA
[root@hadoop01 ~]# hadoop fs -cat /root/outputdata4/000002_0
95010□孔小涛□男□19□CS
95014□王小丽□女□19□CS
95018□王一□女□19□IS
95002□刘晨□女□19□IS
95022□郑明□男□20□MA
95006□孙庆□男□23□CS
[root@hadoop01 ~]# hadoop fs -cat /root/outputdata4/000001_0
95021□周二□男□17□MA
95017□王风娟□女□18□IS
95009□梦圆圆□女□18□MA
95005□刘刚□男□18□MA
95001□李勇□男□20□CS
95013□冯伟□男□21□CS
```

这是分桶操作，按照 id 分桶，并且按照 id 排序

insert overwrite directory '/root/outputdata3' select * from mingxing2 cluster by id;

```
[root@hadoop01 ~]# hadoop fs -cat /root/outputdata3/000003_0
95003□王敏□女□22□MA
95007□易思玲□女□19□MA
95011□包小柏□男□18□MA
95015□王君□男□18□MA
95019□邢小丽□女□19□IS
[root@hadoop01 ~]# hadoop fs -cat /root/outputdata4/000003_0
95015□王君□男□18□MA
95011□包小柏□男□18□MA
95019□邢小丽□女□19□IS
95007□易思玲□女□19□MA
95003□王敏□女□22□MA
```

分桶查询：

指定开启分桶：

set hive.enforce.bucketing = true;

指定 reducetask 数量，也就是指定桶的数量

set mapreduce.job.reduces=4;

insert overwrite directory '/root/outputdata3' select * from mingxing2 cluster by id;

```
hive> insert overwrite directory '/root/outputdata3' select * from mingxing2 cluster by id
;
Query ID = root_20161114191654_266ccdcd-eab5-477e-867c-72d15650948f
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Defaulting to jobconf value of: 4
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1479199464530_0010, Tracking URL = http://hadoop01:8088/proxy/applicati
on_1479199464530_0010/
Kill Command = /root/apps/hadoop-2.6.4/bin/hadoop job  -kill job_1479199464530_0010
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 4
2016-11-14 19:17:04,906 Stage-1 map = 0%,  reduce = 0%
2016-11-14 19:17:13,385 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.25 sec
2016-11-14 19:17:28,062 Stage-1 map = 100%,  reduce = 50%, Cumulative CPU 6.21 sec
2016-11-14 19:17:33,272 Stage-1 map = 100%,  reduce = 75%, Cumulative CPU 7.94 sec
2016-11-14 19:17:34,339 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 9.63 sec
MapReduce Total cumulative CPU time: 9 seconds 630 msec
Ended Job = job_1479199464530_0010
Moving data to: /root/outputdata3
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 4   Cumulative CPU: 9.63 sec   HDFS Read: 16325 HDFS Write:
 527 SUCCESS
Total MapReduce CPU Time Spent: 9 seconds 630 msec
OK
Time taken: 41.481 seconds
hive>
```

3、 按学生名称汇总学生年龄

select name, sum(age) from student group by name;

```
hive> select name ,sum(age) from student group by name;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Defaulting to jobconf value of: 4
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_1388055845553_0055, Tracking URL = http://cloud001:8088/proxy/application_1388055845553_0055/
Kill Command = /root/app/hadoop-2.2.0/bin/hadoop job  -kill job_1388055845553_0055
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 4
2014-01-01 11:05:04,747 Stage-1 map = 0%,  reduce = 0%
2014-01-01 11:05:11,998 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2014-01-01 11:05:13,319 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2014-01-01 11:05:14,865 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2014-01-01 11:05:15,979 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2014-01-01 11:05:17,124 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2014-01-01 11:05:18,218 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2014-01-01 11:05:19,334 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 0.94 sec
2014-01-01 11:05:20,975 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 2.06 sec
2014-01-01 11:05:22,158 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 2.06 sec
2014-01-01 11:05:23,301 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 2.06 sec
2014-01-01 11:05:24,532 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 2.06 sec
2014-01-01 11:05:25,673 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 2.06 sec
2014-01-01 11:05:26,846 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 2.06 sec
2014-01-01 11:05:28,128 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 2.06 sec
2014-01-01 11:05:29,302 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 2.06 sec
2014-01-01 11:05:30,503 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 2.06 sec
2014-01-01 11:05:31,702 Stage-1 map = 100%,  reduce = 25%, Cumulative CPU 2.06 sec
2014-01-01 11:05:33,456 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.26 sec
2014-01-01 11:05:35,005 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.26 sec
2014-01-01 11:05:36,647 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 6.26 sec
MapReduce Total cumulative CPU time: 6 seconds 260 msec
Ended Job = job_1388055845553_0055
MapReduce Jobs Launched:
Job 0: Map: 1  Reduce: 4   Cumulative CPU: 6.26 sec   HDFS Read: 335 HDFS Write: 65 SUCCESS
Total MapReduce CPU Time Spent: 6 seconds 260 msec
OK
ljz     21
mac     18
wp      45
cds     19
```

总结：

一、解释三个执行参数

In order to change the average load for a reducer (in bytes):

set **hive.exec.reducers.bytes.per.reducer**=<number>

In order to limit the maximum number of reducers:

set **hive.exec.reducers.max**=<number>

In order to set a constant number of reducers:

set **mapreduce.job.reduces**=<number>

```
hive> set hive.exec.reducers.bytes.per.reducer;
hive.exec.reducers.bytes.per.reducer=256000000
hive> set hive.exec.reducers.max;
hive.exec.reducers.max=1009
hive> set mapreduce.job.reduces;
mapreduce.job.reduces=4
```

1、直接使用不带设置值得时候是可以查看到这个参数的默认值：

set hive.exec.reducers.bytes.per.reducer

hive.exec.reducers.bytes.per.reducer：一个 hive，就相当于一次 hive 查询中，每一个 reduce 任务它处理的平均数据量

如果要改变值，我们使用这种方式：

set hive.exec.reducers.bytes.per.reducer=51200000

2、查看设置的最大 reducetask 数量

set hive.exec.reducers.max

hive.exec.reducers.max：一次 hive 查询中，最多使用的 reduce task 的数量

我们可以这样使用去改变这个值：

set hive.exec.reducers.max = 20

3、查看设置的一个 reducetask 常量数量

set mapreduce.job.reduces

mapreduce.job.reduces：我们设置的 reducetask 数量

二、HQL 是否被转换成 MR 的问题

前面说过，HQL 语句会被转换成 MapReduce 程序执行，但是上面的例子可以看出部分 HQL 语句并不会转换成 MapReduce，那么什么情况下可以避免转换呢？

1、select * from student; // 简单读取表中文件数据时不会

2、where 过滤条件中只是分区字段时不会转换成 MapReduce

3、set hive.exec.mode.local.auto=true; // hive 会尝试使用本地模式执行

否则，其他情况都会被转换成 MapReduce 程序执行

2.5、Hive Join 查询

语法结构：

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

Hive 支持等值连接（equality join）、外连接（outer join）和（left/right join）。Hive 不支持非

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。

写查询时要注意以下几点：

1、 只支持等值链接，支持 and，不支持 or

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。

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)

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。

3、 Join 时，每次 map/reduce 任务的逻辑

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。

4、 HiveJoin 分三种：inner join, outer join, semi join

其中：outer join 包括 left join，right join 和 full outer join,主要用来处理 join 中空记录的情况

1、 创建两张表：

```
create table tablea (id int, name string) row format delimited fields terminated by ',';
create table tableb (id int, age int) row format delimited fields terminated by ',';
```

```
hadoop01 × hadoop02 hadoop02 (1) hadoop03 hadoop04
0: jdbc:hive2://hadoop02:10000> create table tablea (id int, name string) row format delimited fields terminated by ',';
No rows affected (0.135 seconds)
0: jdbc:hive2://hadoop02:10000> create table tableb (id int, age int) row format delimited fields terminated by ',';
No rows affected (0.145 seconds)
0: jdbc:hive2://hadoop02:10000> show tables;
+-----------+--+
| tab_name  |
+-----------+--+
| tablea    |
| tableb    |
+-----------+--+
2 rows selected (0.084 seconds)
0: jdbc:hive2://hadoop02:10000>
```

2、 准备数据：

先准备两份数据，例如

```
[root@hadoop01 hivedata]# cat a.txt
1,huangbo
2,xuzheng
4,wangbaoqiang
6,huangxiaoming
7,fengjie
10,liudehua
```

```
[root@hadoop02 hivedata]# cat b.txt
2,20
4,50
7,80
10,22
12,33
15,44
```

3、 分别导入数据 a.txt 到 tablea 表，b.txt 到 tableb 表

```
0: jdbc:hive2://hadoop02:10000> select * from tablea;
+------------+----------------+--+
| tablea.id  |  tablea.name   |
+------------+----------------+--+
| 1          | huangbo        |
| 2          | xuzheng        |
| 4          | wangbaoqiang   |
| 6          | huangxiaoming  |
| 7          | fengjie        |
| 10         | liudehua       |
+------------+----------------+--+
```

```
0: jdbc:hive2://hadoop02:10000> select * from tableb;
+------------+-------------+--+
| tableb.id  | tableb.age  |
+------------+-------------+--+
| 2          | 20          |
| 4          | 50          |
| 7          | 80          |
| 10         | 22          |
+------------+-------------+--+
```

4、 数据准备完毕

load data local inpath '/home/hadoop/a.txt' into table tablea;

load data local inpath '/home/hadoop/b.txt' into table tableb;

5、 Join 演示

a)、inner join（内连接）（把符合两边连接条件的数据查询出来）

select * from tablea a inner join tableb b on a.id=b.id;

```
+-------+---------------+-------+--------+--+
| a.id  |    a.name     | b.id  | b.age  |
+-------+---------------+-------+--------+--+
| 2     | xuzheng       | 2     | 20     |
| 4     | wangbaoqiang  | 4     | 50     |
| 7     | fengjie       | 7     | 80     |
| 10    | liudehua      | 10    | 22     |
+-------+---------------+-------+--------+--+
```

b)、left join（左连接，等同于 left outer join）

1、以左表数据为匹配标准，左大右小

2、匹配不上的就是 null

3、返回的数据条数与左表相同

HQL 语句：select * from tablea a left join tableb b on a.id=b.id;

a.id	a.name	b.id	b.age
1	huangbo	NULL	NULL
2	xuzheng	2	20
4	wangbaoqiang	4	50
6	huangxiaoming	NULL	NULL
7	fengjie	7	80
10	liudehua	10	22

c)、right join（右连接，等同于 right outer join）

1、以右表数据为匹配标准，左小右大

2、匹配不上的就是 null

3、返回的数据条数与右表相同

HQL 语句：select * from tablea a right join tableb b on a.id=b.id;

a.id	a.name	b.id	b.age
2	xuzheng	2	20
4	wangbaoqiang	4	50
7	fengjie	7	80
10	liudehua	10	22
NULL	NULL	12	33
NULL	NULL	15	44

e)、left semi join（左半连接）（因为 hive 不支持 in/exists 操作（1.2.1 版本的 hive 支持 in 的操作），所以用该操作实现，并且是 in/exists 的高效实现）

select * from tablea a left semi join tableb b on a.id=b.id;

a.id	a.name
2	xuzheng
4	wangbaoqiang
7	fengjie
10	liudehua

f)、full outer join（完全外链接）

select * from tablea a full outer join tableb b on a.id=b.id;

a.id	a.name	b.id	b.age
1	huangbo	NULL	NULL
2	xuzheng	2	20
4	wangbaoqiang	4	50
6	huangxiaoming	NULL	NULL
7	fengjie	7	80
10	liudehua	10	22
NULL	NULL	12	33
NULL	NULL	15	44